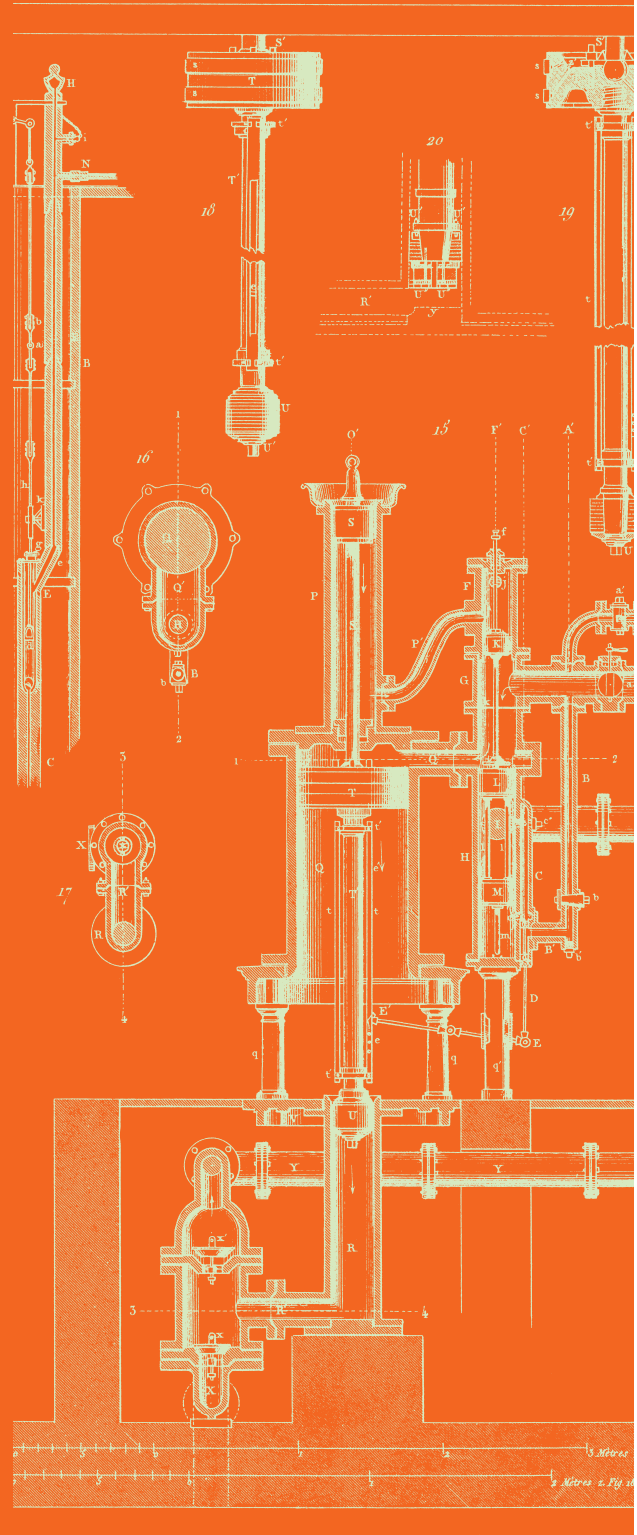
DECEMBER 2006 VOLUME 31 NUMBER 6

# ;login:

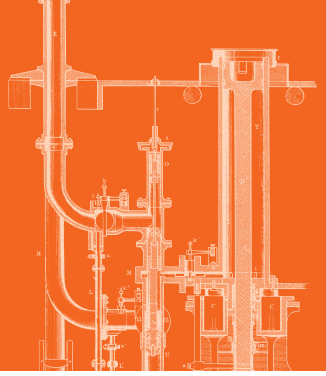
THE USENIX MAGAZINE





USENIX
The Advanced Computing Systems Association

# USENIX Upcoming Events



## 2007 Linux Storage & Filesystem Workshop

Co-located with FAST '07

**FEBRUARY 12–13, 2007, San Jose, CA, USA**

**http://www.usenix.org/lsf07**

## 5th USENIX Conference on File and Storage Technologies (FAST '07)

Sponsored by USENIX in cooperation with ACM SIGOPS, IEEE Mass Storage Systems Technical Committee, and IEEE TCOS

**FEBRUARY 13–16, 2007, San Jose, CA, USA**

**http://www.usenix.org/fast07**

## 1st Symposium on Computer Human Interaction for Management of Information Technology (CHIMIT '07)

Sponsored by ACM in cooperation with USENIX

**MARCH 30–31, 2007, Cambridge, MA, USA**

**http://chimit.cs.tufts.edu**

## Second Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML07)

Co-located with NSDI '07

**APRIL 10, 2007, Cambridge, MA, USA**

**http://www.cs.duke.edu/nicl/sysml07**

## Third International Workshop on Networking Meets Databases (NetDB '07)

Co-located with NSDI '07
Sponsored by USENIX in cooperation with ACM SIGCOMM

**APRIL 10, 2007, Cambridge, MA, USA**

**http://www.usenix.org/netdb07**
Paper submissions due: January 12, 2007

## First Workshop on Hot Topics in Understanding Botnets (HotBots '07)

Co-located with NSDI '07

**APRIL 10, 2007, Cambridge, MA, USA**

**http://www.usenix.org/hotbots07**
Paper submissions due: February 26, 2007

## 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI '07)

Sponsored by USENIX in cooperation with ACM SIGCOMM and ACM SIGOPS

**APRIL 11–13, 2007, Cambridge, MA, USA**

**http://www.usenix.org/nsdi07**

## 11th Workshop on Hot Topics in Operating Systems (HotOS XI)

Sponsored by USENIX in cooperation with the IEEE Technical Committee on Operating Systems (TCOS)

**MAY 7–9, 2007, San Diego, CA, USA**

**http://www.usenix.org/hotos07**
Paper submissions due: January 4, 2007

## 5th ACM/USENIX International Conference on Mobile Computing Systems, Applications, and Services (MobiSys 2007)

Jointly sponsored by USENIX and ACM SIGMOBILE, in cooperation with ACM SIGOPS

**JUNE 11–15, 2007, Puerto Rico**

**http://www.sigmobile.org/mobisys/2007/**

## Workshop on Experimental Computer Science (ECS '07)

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

**JUNE 13–14, 2007, San Diego, CA, USA**

**http://www.expcs.org/**
Paper submissions due: February 23, 2007

## Third International ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments (VEE '07)

Sponsored by ACM SIGPLAN and ACM SIGOPS in cooperation with USENIX

**JUNE 13–15, 2007, San Diego, CA, USA**

**http://vee07.cs.ucsb.edu**
Paper submissions due: February 5, 2007

## 2007 USENIX Annual Technical Conference

**JUNE 17–22, 2007, Santa Clara, CA, USA**

**http://www.usenix.org/usenix07**
Paper submissions due: January 9, 2007

## 16th USENIX Security Symposium

**AUGUST 6–10, 2007, Boston, MA, USA**

**http://www.usenix.org/sec07**
Paper submissions due: February 1, 2007

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

# contents




;login:

**VOL. 31, #6, DECEMBER 2006**

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
David Couzens
proofshop@usenix.org

PRODUCTION
Lisa Camp de Avalos
Casey Henderson

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.

$85 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $115 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@usenix.org*

**I RECENTLY READ A SUMMARY OF** new vulnerabilities in 2006 which stated that, for the first time, buffer overflows had fallen from number one to number four. The new "leaders" in the vulnerability and exploit race were cross-site scripting (XSS), SQL-injection, and PHP file inclusion. Many pundits drew from this the conclusion that buffer overflow vulnerabilities are in decline. If only that were true, perhaps we would be seeing some light at the end of this tunnel.

Developers, security companies, and vendors have done considerable work to retire buffer overflows as security issues. The early work includes scripts that simply scan for offending string manipulation functions, such as strcpy() in source code. Replacing strcpy() with strncpy() adds bounds checking when copying a string, and as long as the length of the destination string is used as the bounding value, this actually helps a lot. In the security business, we call this "low-hanging fruit," as finding and fixing these vulnerabilities is as simple as running a tool.

Other pathways to uncovering buffer overflows include tracing input flows. An attacker manipulates buffer overflows by providing some malicious input, so the defending coder needs to follow the flow and see how any input gets used. I don't want to make this sound easy, because it's not. Take the Sendmail vulnerability in the address checking function that missed decrementing a counter while parsing email addresses. That bug had been there for many years, unnoticed—until it was found in 2003. A second, similar bug was found later in 2003, also in code that checks email addresses.

Code reviews that can find bugs like these are tedious. After the first bug was found, I downloaded the patches for Sendmail, then used them to find the buggy routine (although there were multiple functions patched, looking for ones related to user input helped to shorten the search). Then I walked through the crackaddr() function in the header.c file, certain the mistake would be found there. The error was there, I guessed correctly how it worked, but I didn't actually uncover the crucial place in this function where a variable should have been decremented. It just isn't that easy.

A hacking group didn't find the missing decrement either. They debugged vulnerable versions of Sendmail and found a way to overwrite a FILE object so that a read callback would point to their own, handcrafted code. This code makes an outgoing connection to port 25/tcp at the IP address of the attacker's choosing [1] and runs a root-owned shell.

## Invoking the Hardware Mantra

Since code reviews are so difficult, coming up with a solution that bypasses code reviews is imperative. And there are many such solutions, some done in software, and better ones manifested through appropriate use of hardware.

Crispin Cowan et al. [2] created the idea of placing a value in memory that would be overwritten during a buffer overflow, and testing that value works as a check for buffer overflows. This idea, known as the "stack canary," forms the basis for many software protection schemes, including one used by Microsoft. If this worked reliably, there would no longer be exploitable buffer overflows in Microsoft code, as evidenced by the emergency patch released on September 26 to fix a buffer overflow in the Vector Markup Language (vgx.dll) code in IE versions 5 through 7 [3].

There are, of course, other techniques that can be used to thwart buffer overflows. RedHat, in its versions of the Linux kernel, uses several software techniques, most notably by changing the layout of memory during process creation. By changing the location of the stack within an 8K window, getting an exploit to work correctly becomes much more difficult (but could succeed eventually).

Sun Microsystems has long included software support in its kernel for hardware protection. Sun's SPARC chip architecture makes it easy to make the stack nonexecutable, and this has been an option since the mid-1990s in Solaris. Other prominent CPU vendors lagged way behind on this feature. Intel processors, until more recent versions, could have nonexecutable stacks only by using a kludge that involved segment registers. New AMD and Intel processors now make it much simpler to enable hardware stack protection by no longer lumping write and execution permission bits in memory management together.

## Kiss of Death

Even if programmers and chip designers had solved stack-based buffer overflows, we still have to deal with other related programmer errors, including format string, integer overflow, and double free bugs. None of these ever attained the number 1 status of buffer overflows, but all make it into the top 40 vulnerabilities [4]. Protecting the stack does make exploitation more difficult, but not impossible, as people are still writing both buffer overflow and other bug-related exploits.

When I posted some of my thoughts about buffer overflows being toppled from number 1 by Web scripting bugs, I got an even more thoughtful response from Chris Wysopal. Chris pointed out that a buffer overflow that gets prevented by *any* of the methods I've mentioned here has been converted into a Denial of Service attack. All the mechanisms designed to defend against buffer overflow to date halt the execution of the offending program. With the exception of multiply threaded servers, such as Apache, or ones that watch for untimely server death, such as Postfix, the server

has died, denying service. Remember, if your Web browser mysteriously dies while following a link, it may be a victim of an "unsuccessful" buffer overflow attack.

"We stopped the buffer overflow attack, but the patient died." How sad.

Just as significant, Chris pointed out that buffer overflow attacks *have not declined statistically*. To me, this is the most damning point of all. Buffer overflows have fallen to number 4 in the top 40 CVE vulnerability list only because Web scripting bugs have increased in popularity. The absolute number of buffer overflows has remained almost constant over the past five years.

## Web Services

Web scripting/programming errors now make up four of the top five reported vulnerabilities. The "dot" category stands for vulnerabilities that rely on the use of ".." to view or execute files that should be protected. I would have thought that, like SQL-injection and XSS, these attacks should have fallen to careful inspection of client-supplied input. Sadly, I am mistaken. People continue to make the same mistakes year after year.

I don't want to trivialize this problem. XSS, in particular, is difficult to deal with, although scrubbing <script> from user input to Web scripts would go a long way toward curing this issue.

What really struck me was the "new" PHP bug, file inclusion, involving simply appending a bit of text to a request to a PHP script. PHP has a reputation for making it easy to write insecure Web scripts, and this flaw certainly bolsters that impression.

## The Lineup

Team CYMRU leads off the 2006 Security focus issue with an article about the computer underground. Rob Thomas gave the keynote at SRUTI [5] on this very topic. If you have ever worried about sharing your personal information online, or with anyone, whether he or she is a bank clerk or just some anonymous person on the phone, you will want to read this article. It's not just the level of fraud but, as the authors write, the lack of any attempt to hide criminal activities that is just astounding. No wonder we haven't caught bin Laden by this time if the United States and other countries continue to permit blatant identity trading to go on (not that identity traders are never caught; see [6]).

Next, a trio of German researchers share information about how they detect and quarantine infected Windows systems on a university network. Through the use of Nepenthe, a low-interaction honeypot, they can both capture malware and detect without any false positives infected systems on their network. I know that this is an issue for many organizations, and the approach described here appears worth pursuing. Also, Göbel, Hektor, and Holz describe the Haxdoor malware and the vast cache of identity information it captured in just nine days.

I so enjoyed Andy Ozment's Security '06 presentation that I asked him to write for *;login:*. Andy and Stuart Schechter have statistically analyzed the bugs found in OpenBSD, relating each bug not just to the code patched but to when that code appears in OpenBSD. Perhaps there is some light at the end of this security tunnel after all.

Nick Weaver and Dan Ellis carefully explain why white worms, those that attempt good works rather than exploits, are not a good idea. If you have ever considered writing a white worm, this article is certain to dissuade you.

Mike Scher, past *;login:* contributor and winner of the "legal counsel of this issue" award, entertains us with musings about tort law and negligence. A recent case grabbed Mike's attention, and he uses a similar, but fictitious, case to describe just how an organization might be liable for negligence. If you write policy or have anything to do with administering public servers, you need to read this article.

Finally, Mike Howard argues that changing passwords too often can be more harmful than simply using strong passwords. I believe Mike makes his case well.

In the Sysadmin section, Mark Burgess continues his excellent series about the nature of configuration management. After reading (and editing) Anderson's "Configuration Management" [7], I thought I knew it all. But Mark has a different way of viewing things and a very convincing way of getting the reader to look at configuration management issues in a new light.

Dave Josephsen's article borders on security, but it really does belong in the Sysadmin section. Dave got me interested when he told me about BGP hijacking attacks designed to support spammers. Stopping spam is Dave's real focus, but the BGP attack is interesting in itself. Dave narrates the story of the spam wars, reaching a conclusion you might recognize if you have read Dave's writing before.

Our columnists have done their work as well, with David Blank-Edelman deciding to write about security in the same sense that TSA protects those flying in U.S. airspace. Robert Haskins writes about the use of wireless by ISPs, including solutions that may apply to those who have chosen to live remotely. Heison Chak considers the security-related aspects of using VoIP. Robert Ferrell enlightens us on the uses of sledgehammers in computer security.

In the Book Reviews section, Elizabeth Zwicky continues her search for good Windows security books, finding two, and tells us about reading *Security and Usability*, an interesting collection of papers that came out last year. Sam Stover reports on yet another Syngress book that contains some new material and loads of repeated chapters. Finally, I managed to write a couple of reviews myself.

Nick Stoughton has produced another article in his series about standards. Nick has been the USENIX representative for standards for many years, and these articles allow you to get an insider's view of standards processes (some of which will affect you).

We have three summaries in this issue, all related to security. The Security '06 conference summaries belong in this issue, of course. Dan Geer has produced excerpts from a much longer summary of MetriCon, the first workshop on security metrics. And the organizers of the New Security Paradigms Workshop have produced a very concise summary of hours of discussion.

I would like to leave you with some parting thoughts related to the CYMRU article about stolen identity. During my Guru presentation in Boston at USENIX Annual Tech, I pondered aloud about how it is that in this supposedly modern age our identity can be described in approximately 160 bytes of information. This small amount of data covers everything you would need to present to get a car or home loan—and every-

thing an identity thief would need as well. I think this is absurd, but I will confess that I haven't come up with (and patented) a workable solution.

In the meantime, I suggest that you watch your own credit and banking reports closely. Someone else's jackpot could easily be your own misfortune.

## REFERENCES


[1] "Technical Analysis of the Remote Sendmail Vulnerability": http://lwn.net/Articles/24292/.

[2] "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks": http://www.usenix.org/publications/library/proceedings/sec98/cowan.html.

[3] "Microsoft Internet Explorer VML Buffer Overflow": http://www.kb.cert.org/vuls/id/416092.

[4] "Vulnerability Type Distribution in CVE," posting about the CVE top 38 (not 40, but that sounds better) vulnerabilities of 2006: http://www.attrition.org/pipermail/vim/2006-September/001032.html.

[5] 2nd Workshop on Steps on Reducing Unwanted Traffic on the Internet: http://www.usenix.org/events/sruti06/.

[6] "Hacker Hunters," *Business Week* (May 30, 2005): http://www.businessweek.com/print/magazine/content/05_22/b3935001_mz001.htm?chan=tc.

[7] P. Anderson, "System Configuration": http://www.sage.org/pubs/14_sysconfig/.

TEAM CYMRU

# the underground economy: priceless

Rob Thomas is a long-time network security professional and founder of Team Cymru. He has written many papers on information security and spoken at numerous conferences worldwide on the topic of Internet security.

*robt@cymru.com*

Jerry Martin is an advocate of the complete information assurance process: risk assessment, policy development, solution deployment, and user education. He has worked in several information security positions, including at the U.S. Air Force.

*jerry@cymru.com*

**THE CYBER UNDERGROUND ECONOMY** is just as seedy and illegal as its physical counterpart. The primary objective of those who operate there is *money*. The National Cyber Security Alliance published some data a while ago that concisely describes the problem:

1. Fully 61% of U.S. computers are infected with spyware.
2. Americans say they lost more than US$336 million last year to online fraud.

These figures are largely based on self-reporting, which is often suspect. Given the enormous quantity of data witnessed on numerous Internet Relay Chat (IRC) channels, both numbers may be underreported. Given these staggering numbers, one might well ask what is being done to address this criminal activity. Lamentably, the answer is, "Not much." The popular school of thought is that finding and prosecuting these perpetrators of financial fraud and outright theft is too costly, too resource-intensive, and just too hard. This article will expose the infrastructure the miscreants have established; the open arrogance the buyers, sellers, traders, and cashiers exhibit; the activities and alliances in which the underground denizens are involved; the method by which they receive their ill-gotten goods; the blatant manner in which they advertise; and the personal data that is harvested every single hour of every day of the year. Numerous snippets of captured IRC chatter will illustrate the points raised, although the nicknames and the information harvested are obfuscated.

The miscreants can make a handsome living through these activities. Even those without great skills can barter their way into large quantities of money they would never earn in the physical world. It is important to note that these miscreants are located all over the globe, and thus they may be earning well above the average income for their areas.

## Infrastructure

Entire IRC networks—*networks*, not just single servers—are dedicated to the underground economy. There are 35 to 40 particularly active servers, all of which are easy to find. Furthermore, IRC isn't the only Internet vehicle they use. Other conduits include, but are not limited to, HTTP, Instant Messaging, and Peer-to-Peer (P2P).

Increasingly, many of the miscreants utilize encryption in these services, such as VPNs or SSL. The following table illustrates the number of cards compromised in three months for a single server!

| *Month* | *Amex* | *Visa* | *MasterCard* | *Discover* |
|---|---|---|---|---|
| 2005/10 | 70 | 28942 | 11820 | 1064 |
| 2005/11 | 51 | 31932 | 13218 | 1214 |
| 2005/12 | 89 | 26492 | 10662 | 1079 |

The miscreants in the underground economy are typically self-policing. Each IRC network will normally have a channel, such as #help or #rippers, dedicated to the reporting of those who are known to conduct fraudulent deals. The operators of these networks will ban the nicknames of those who have a proven history of fraud. This is a form of self-regulation that ensures the sellers and buyers have a "pleasant" experience and attempts to elicit repeat visits. The miscreants keep meticulous records of those who have defrauded them, and they are quick to share those records with everyone. As with all criminal societies, there is a fair amount of fraudulent dealings and "ripping" (bad business deals).

The tale goes something like this: Miscreant <A> advertises a need for roots, which are compromised UNIX systems on which someone has obtained root access. <B> disappears for a while to have a private conversation with <A>, which is the norm for those finalizing deals. <B> then pastes that conversation into the open trading channel as a warning to other miscreants:

<B> i rember when u tried to sell me a root scanner
<B> lol were u going to try scam me
<B> yeah
<B> coz u told me last weekk u had a private root scanner
<A> i need it
<B> you were going to try scam me
<B> A is a scammer so beware
<B> 1 day he trys selling me a root scanner next day he needs roots
<B> so beware

Rest assured that a great many miscreants will now avoid conducting business with <A>. In contrast, some miscreants are well vetted, with certain underground economy networks and forums bestowing verification on those miscreant merchants who are considered reliable.

Because these miscreants encounter a wide variety of fraud, they capitalize on the existing criminal support structure, which includes places to obscure their ill-gotten profits. "OS" and "os" mean "offshore bank" in this case:

<A> whats a good os bank?
<B> you can use webmoney
<B> if you can deal with their fees
<B> its not a OS bank
<B> but they wont ever freeze your account
<A> .ee right?
<B> no
<B> thats an exchanger
<B> `www.wmtransfer.com`
<B> is the official webmoney site

## Buyers, Sellers, and Traders

Even the miscreants who spend most of their days herding bots have requirements, mostly in support of their profit-making activities with those bots. In one bit of chatter, a miscreant is specifically seeking an IRC daemon to prevent interlopers from running the intelligence-gathering commands. Such IRC daemons do exist, as do the IRC daemons that produce bogus responses. The needs and those who meet them do have exchange rates, though these can fluctuate wildly. Some miscreants willingly list their prices, such as <A> in the following:

> <A> Sell Cvv US(1$ each),Uk(2$ each)Cvv with SSN & DL(10$ each)and ePassporte Account with 560$ in acc(50$),Hacked Host(7$),Tut Scam CC Full in VP-ASP Shop(10$).shopadmin with 4100 order(200$), Tool Calculate Drive Licsence Number(10$).... I'm sleeping. MSG me and I will reply U as soon as I can !

Compromised hosts sold by <A> cost US$7, which is quite a lot, considering that the average bot costs US$0.04 (four cents). Note as well that <A> is actually asleep; his scripted advertisements continue unabated, providing a steady marketing opportunity.

Dealing in the underground economy is not without risks, however, and the merchants realize this fact. This is why they provide friendly advice designed to maintain the safety of those who visit their servers. <A> is a bot that emits this message at regular intervals:

> <A> Precautions 1. Use Fake Ip or Use a VPN While On This Server. 2. Do Not Use Your Real ID in picking any type of money 3. Dont give your real information to anyone unless you know him/her. 4. keep your self safe on [ UNDERGROUND ECONOMY IRC NETWORK ]. ThankS!!

One can readily see the plethora of advertisements by the miscreant merchants and the miscreant consumers regarding compromised financial accounts, drops (compromised financial accounts used to launder funds), and cashiers (those who can clean them out):

> <A> i have wells and boa logins and i need to good drop man .....ripper f#@! off
> <B> <=== .Have All Bank Infos. US/Canada/ Uk ...Legit Cashiers Only Msg/me
> <C> HELLO room... I am Ashley from the State... I got drops for US banks and i need a very trust worthy and understanding man to do deal with ... the share its 60/40...Msg me for deal

The miscreant spammers are some of the most highly paid individuals in the underground. It's easy to see why—spam works, and yields high profits. In one particular instance, the spam involved online (illegal) diplomas. At US$1000 per diploma, it's obvious why these sponsors can afford to pay the miscreant spammers. The miscreant spammers can then afford to pay the proxy creators, malware creators, etc.

## Activities and Alliances

The miscreants have all sorts of capabilities and all sorts of needs. They aren't so unimaginative as to limit their opportunities. When they have needs, they offer remuneration or even partnership. One miscreant advertises his need for a spam mailer to fill Hotmail inboxes. He is willing to pay through e-gold, or will even share the proceeds from his phishing (the miscreants call it "scam") site. They'll happily pay for that which no one

will trade. Selling source code for malware (rarely 0days) is another avenue to profit. It is critical to note that these aren't the malware authors. There are miscreants who spend quality time obtaining source code and reselling it. Some of them, a very few, will also modify it for a fee.

Those who provide services in the underground economy are looking for long-term customers. It pays to keep the customers happy and to nurture those illicit business relationships. Oh, and don't believe those reports from security vendors who say all hacking happens in South Korea, or Brazil, or China. There are miscreants everywhere (e.g., <A> i need any hacker from 'country x').

The large number of online merchants that are regularly hacked is both sad and impressive:

> <A> selling admin database password of hacked online store with hundreds of cvv2 and check draft payment (it's Bank accounts # and Routing#) inside. I receive the payment 1st (WU/E-Gold). Also trade cvv2 for [WEB SITE] account.

Carding servers is easy, thanks to the proliferation of hosting sites that provide online purchase and configuration capabilities. The good news for the miscreants is that they no longer rely on boxes on their cable or DSL links. They now have professional, redundant hosting for their IRC servers, botnet servers, etc. Even exclusive, rare credit cards will be stolen. One can just imagine the purchasing power <A> has with this card:

> <A> I got an american express black card the other day
> <A> weird huh?
> <B> ... black card?
> <B> i thought it wa blue
> <A> go look it up
> <A> its called the centurion
> <B> first link has "black is beautiful" in the thingy
> <B> and it's talking about the card

It is also a reality that miscreants actually buy physical goods in the underground economy:

> <A> Sell cc's full info with PIN (debit, credit), COB's Laptops (alienware area51 = 500$, Dell inspiron 6100=400$, Scam pages (ebay, aol, paypal, egold, escrow, earthlink), track2gen (.exE) support 857 bins, 2000 bins (update bins), root. Payment (wu or e-gold).

The miscreants are avid proponents of online banking, particularly other people's online bank accounts. This has been and remains quite the popular activity, with accounts compromised daily. These accounts may be traded several times prior to any activity passing through them. Buying and selling compromised bank accounts continues unabated. Email inboxes, with their lack of security, provide yet another compromise vector for personal information:

> <A> has everything on 1 person from there emails,to paypal to ebay, online banking .travelcity , expedia , you name it i probably have it full info cc high limit /msg me

One miscreant inquires about the worth of 40K compromised financial accounts. It isn't worth much to him because he cannot cash it out. For this task, he requires a cashier and a drop (a place or account through which to route the money to him). For this reason, he will be paid pennies on the dollar for his collection of compromised financial accounts, which is just fine:

<A> how much would a lets say 40k
<B> with all informations 40k ??
<B> Fulls
<A> user name and pass
<A> 200-300 an account ?
<B> variable between 250 $ =====> 500 $
<A> ill retire in a month

This is the greatest failure of new technology—a rush to market, without consideration of the risks and a cost/benefit analysis. This is at the heart of the security problem. Certainly, that is not to say that industries should not capitalize on technological advances but, rather, that they should consider risk and threat mitigation strategies prior to bringing any product to market.

Obtaining a fake ID to cash out an illegal funds movement is an easy task in the underground. Many of the miscreants sell or buy such IDs, or teach others how to create them. There is quite the cottage industry providing supplies for this endeavor. All of the ID chatter is for U.S. IDs of various sorts, including college IDs, state IDs, and drivers' licenses.

## Cashiers

Extracting cash from the underground economy is the goal of many, if not most, participating miscreants. They find all sorts of ways to accomplish this goal, though these aren't new techniques; physical world criminals have been doing this for years. So what's different? Online crime is often easier and has a lot less inherent risk. The biggest challenges to the miscreants aren't IDS, firewalls, 0day creation, or any other technological hurdle. The biggest challenge is where to cash the checks. Those who actively participate in the underground economy have another problem—how to move the significant quantity of illegally obtained funds. There are a variety of solutions they discuss, such as offshore trusts to protect their financial assets against lawsuits. Lawsuits, prying eyes, and seizure are all mitigated through the use of offshore banking. Several offshore banks will wittingly accept such accounts.

The miscreants advertise for cashiers for both logical and physical (e.g., go collect the money at a Western Union site) account cleanups. Cashing out these accounts often must be accomplished from within the country where the account resides. Enter the bank broker, the miscreant who will cash out the account. Demand is high for these miscreants, and they never ask questions. When a cashier attempts to clean out a bank account (50% always goes to the cashier) on behalf of another miscreant, that cashier must have some semblance of legitimacy with the bank. Increasingly, the miscreants are finding that a male voice attempting to clean out an account obviously belonging to a female isn't accepted by the banks. Thus is born a new skill set: gender-based cashiers. There are plenty of female miscreants, willing to clean out accounts both virtually and physically. When the market makes a demand, the demand-based underground economy responds:

<A> i need who can confirmer westernunion female visa
<B> speaking of wu, who can do females?

The miscreants who serve as cashiers in the underground economy are ready and waiting to fill the orders. They are happy to generate cash transfers, often through services such as Western Union. The mule or the intended recipient then picks up the cash at a Western Union office. This is both easy and convenient, the benchmarks of success in the increasingly online world. Although slightly obfuscated, this example is quite real:

<A> Western Union Money Transfer? Pick Up Notification.
<A> Dear X X,
<A> Thank you for using the Western Union Money Transfer
<A> Your money transfer has been picked up by the receiver.
Following is a summary of your transaction.
<A> XXXXXXX508
<A> Date of Order:
<A> 09/15/2005
<A> Amount Sent:
<A> $900.00
<A> Receiver Name:
<A> X X
<A> Status:
<A> Picked Up
<A> write me if u want me to cashout creditcard for you throgh westernunion

It's easy to move money online, because of the large number of cashiers. These criminals widely and loudly advertise their skills, prices, and specialties. They are competing for the business of other miscreants and are certain to add that touch of quality customer service:

<B> I have Bank drops for Quick Cashout in(Hsbc,Wells, Lloyds, Citibank,Boa, Barclays,Woolwich,rbc) Contact me now for Fast Cash out..Deal is 50% each
<D> Hello,I'm a professional MTCn confirmer if you have any order pending you can IM me,i have done so many transaction for different people and also i made different kind of transfer into account such as BAO, WELS,HSBC any body with full infos for this account who wanna transfer should IM me now and also i have BIN,EBAY SCAM PAGES,PHP bulk mailer if anyone is interested IM me all rippers keep off.NOTE I VERIFY FIRST.....

It's an odd use of the term "professional," but, in fact, these *are* skilled, reliable professionals. They know their business, are risk-averse, and are rarely caught. (What is the MTCN? That is the Money Transfer Control Number.)

## Drops

One of the hottest commodities in the underground economy is the drop. A drop can have one of two definitions. The first definition of a drop is a location to which goods or cash can be sent. The person who owns the drop will then resend the items or hold them for pickup. There is a charge for this service, of course, ranging from a 70/30 (30% to the drop owner) split to a 50/50 split. Drops include homes and businesses, and often the drop owner is clueless about the contents of the dropped package. In this case, the drop owner is paid a flat fee by the shipper or the broker. The second definition of a drop is a bank account through which money can be moved. This is a convenient way to cash out bank accounts, online financial accounts such as PayPal, and credit cards. The drop owner almost always receives 50% of the take, although competition in this space is reducing that percentage. The location of the drop is critical, as some companies won't ship overseas. Some miscreants want a drop close to home and physical access. The demand is never-ending, with the greatest demand placed on U.S.-based drops, although some are undesirable. Where there is demand, there shall be supply. The underground economy abhors a vacuum.

There are miscreants who need drops in certain countries or who are able to cash out bank accounts in another country. Sometimes the same miscreant will conduct "business" in several nations. This miscreant is looking for drops (shipping addresses to which fraudulently obtained or outright stolen goods can be delivered) in a number of nations. The list of nations in which <A> will do business is both interesting and impressive:

> <A> I NEED DROPS FOR PHONES AND PDA's in Singapore Australia Austria Belgium Brunei Darussalam Canada China Denmark Finland France Germany Greece Hong Kong Indonesia India Ireland Israel Italy Japan Korea (South) Luxembourg Macau Malaysia Netherlands New Zealand Norway Portugal Saudi Arabia Spain Sweden Switzerland Taiwan Thailand United Arab Emirates United Kingdom United States

Criminals know no boundaries, and online crime is an international business. The miscreants understand ROI, and they also understand that an alliance only needs to last as long as it takes to accomplish the goals. The miscreants continue to build large networks of like-minded criminals. It will take a global network to thwart them. The denizens of the underground economy aren't unlike anyone else. They would like to retire at some point. Unlike honest, hard-working people, however, the miscreants have a paucity of ethics and perhaps a faster path to retirement.

## Advertising

> <A> JOIN #[ CHANNEL ] THE BEST HACKER CHANNEL!!! JOIN US ..!!! U CAN BECOME HACKER AND RICH...!!!!

Doesn't that just about sum it up? Things have changed; these aren't the miscreants who will hack something, get arrested, and land a six-figure job at a security consultancy (though they might sell their sploits to one). Now they'll be paid to hack things, or write tools to hack things, or just sell things they've hacked, and not get arrested, and still make great money.

When a miscreant is dealing in compromised financial accounts, the miscreant must advertise to attract business. This proves to any potential consumers that the miscreant has the goods and can deliver. <A> begins by sharing some data from one of his collections of compromised accounts:

> <A> Account Summary
> <A> For optimal viewing of the Wells Fargo Web site, we recommend that you enable CSS
> <A> Cash Accounts
> <A> Account Account Number Available Balance
> <A> CHECKING 367-3157xxx $425.38
> <A> Total $425.38
> <A> Credit Accounts
> <A> Account Account Number Outstanding
> <A> Balance Available
> <A> Credit
> <A> VISA ( View Spending Report ) xxxx-xxxx-xxxx-9556 -$80.82 $5,900.00
> <A> Total -$80.82 $5,900.00
> <A> To end your session, be sure to Sign Off

It isn't strictly about extracting cash, however. The miscreants use the cash to obtain other goods, or they sell goods for cash. This gives us some sense of the relative value of certain goods. The bulk sales offer is troubling, as it

indicates the result of a larger compromise:

> <D> Selling CVV.$USD 3 EACH.IF BUY IN BULKS(100) 2 USD EACH

The underground shopping mall is open, as always, and happy to provide for every need. Be it through cash or barter, everything is for sale. Many of the traded items can yield hard currency:

> <B> got ebay/paypal/yahoo/hotmail/citibank/aol scams + psyBNC + hotmail mailbox closing exploit + how to crack anything + can create scam of any site in very little time + Track2gen.exe. Got millions of proxies all over the world + 3million ebay/paypal/egold mail list + track2gen(.php)(.exe) + dark-mailer pro v1.9(300$) + gamadye mailer registered(140$) + wolrds fastest mail bomber to get them msg me

## Data Stolen

How much money do the miscreants make in the underground economy? More to the point, how much money do they steal? Here's a snapshot from one underground economy trading channel over a 24-hour period. These are the total account values for financial accounts to which these criminals have obtained access. These are just the *samples*; these miscreants claim to have many more accounts to sell, and they offer up the samples as advertising. All amounts are in U.S. dollars, and some of these account totals are impressive, while others are quite small. The true account owner probably doesn't consider them unimportant, however:

> <A> Total: $310.64—A is from Country A
> <B> Total $930,391.94—B is from Country B
> <C> Total $216,934.93
> <C> Grand Total $1,803.59—C is from Country C
> <D> Total: $49.00—D is from the Country D
> <E> Total $258,602.27—E is from Country E
> <F> Total $60.07—F is from the Country D
> <G> Grand Total $1,987.97—G is from Country F
> <H> Total $48,096.65—H is from Country A
> <I> Total $33,332.76—I is from Country B

So, with one channel, one 24-hour period, and just a few samples, at least US$1,599,335.80 has gone to fund multinational criminals.

When your credit card details are stolen, *all* of the details are stolen. When a miscreant offers up a "full" or "full info" for sale or trade, that miscreant will have the goods. Here is an example from <A>, an overseas miscreant. The victim's details have been slightly obfuscated. There is no such thing as a "secret question" when it comes to the miscreants:

> <A> Name: Jason XXX
> <A> Address 1: XXX S University Blvd.
> <A> City: XXX
> <A> State: OK
> <A> Zip: XXXXX
> <A> Country: usa
> <A> Home Phone: (XXX) XXX-X991 Ext:
> <A> Date Of Birth: 12/8/19XX
> <A> Social Security Number: XXXX32199
> <A> Mothers Maiden Name: Reaves
> <A> Drivers License Number: XXXX24766
> <A> Drivers License State: OK
> <A> Secret Question: What is your pet's name?
> <A> Secret Question Answer: Joad

<A> Name On Card: Jason XXX
<A> Credit Card Number: 4492XXXXXXXX8831
<A> Credit Card Brand: Visa
<A> Credit Card Type: Credit
<A> EXP Date: 4/2006
<A> Credit Card PIN Number:
<A> Card ID Number: X46
<A> Card Bank Name: OU Federal Credit Union
<A> Card 1800 Number: 1800XXXXX9
<A> eBay User ID: XXX
<A> eBay Password: XXXXXX
<A> eBay Password: XXXXXX
<A> **********
<A> **********

Unfortunately, the little snippets don't provide a sense of the frequency of such advertising or the inferred frequency of the transactions between buyer and seller. In one six-minute period of underground economy chatter from one underground economy network and channel, *eight* distinct miscreants sought to launder stolen money. This is fairly typical, with advertisements coming in ebbs and flows, drops and transfers occurring, and boldly advertising cashiers doing deals. Such activities are all readily accessible to anyone and everyone, with numerous participants happy to route money out of individuals' accounts to anywhere at all.

The use of keylogging and data-extraction bots, which is just about every bot now installed, has enabled the miscreants to have ready access to bank accounts and other financial accounts:

<A> selling Bank Of America online access with $10,000 and other with $900 balance. Payment : Western Union
<B> who can cashout Bank Of America/Washington Mutual without pin but with online access msg me and lets make a great deal !
<C> can cashout verified paypals in 2 days. $2000 every couple of days. 75/25. Msg me for deal
<D> Payee: Centennial Bank

One miscreant even provided a screen shot of a compromised Wells Fargo account, with a net total of US$21,431.18 in cash.

## Conclusion

The underground economy is fertile ground for the pursuit (and, we hope) prosecution of the miscreants. Most of the underground economy servers are public, advertised widely, and easy to find (standard IRC ports, very descriptive DNS RRs, etc.). There is absolutely no presumption of privacy in the underground economy; the channels aren't hidden, the channels have no keys, and the servers have no passwords. The clients in these channels are widely divergent. Think about what has just been shared:

1. There is no need for specialized IRC clients.
2. There is no need to rapidly track ever-changing DNS RRs and IPs.
3. There is no need to pull apart every new permutation of malware.
4. There is no need to hide, period.

This is a cosmopolitan mix; there is evidence of physical crime as well as online crime, and admissions of guilt, and all are readily available. Although the data in this article is obfuscated, these stanzas of gross fraud come with the name, address, phone number, SSN, and mother's maiden

name of the victim. That seems ready-made for a complaint and one might imagine that a prosecutor, judge, and jury would understand those blatant advertisements. These same individuals have, in the past, been successfully educated about DDoS, hacking, and warez. Even in child-exploitation cases, the jury learned about the methods by which such horrors were shared online. There are approximately 38 active underground economy IRC servers at present, and most of these are located in the United States.

If more than one year can be expended tracking a pair of bot-herders, then surely logging and tracking the miscreants who are online, advertising their crimes, is worth the resource expenditure. If the goal is putting a dent in online crime, then focus on the biggest perpetrators instead of the insignificant players. It is imperative to hit the miscreants where it hurts—in the conduits for their ill-gotten gains.

It is well past time to use the miscreants' greatest asset, the underground economy, against them. It seems that many people still remain largely unaware of the underground economy. They remain unaware that the underground economy drives most of the Internet-based malfeasance everyone (largely silently) endures. In the end, almost everything comes down to money. Certainly, the stated reasons for an action might be religious or nationalist, but those actions are funded by only one thing—*money*. The underground is a reflection of the real world, and to ignore it is to ignore the real world.

JAN GÖBEL, JENS HEKTOR, AND
THORSTEN HOLZ

# advanced honeypot-based intrusion detection

Jan Göbel has an M.Sc.in computer science from RWTH Aachen University and wrote his diploma thesis on "Advanced Honeynet-based Intrusion Detection." He is currently working at the Center for Computing and Communication at RWTH Aachen.

*goebel@rz.rwth-aachen.de*

Jens Hektor holds an M.Sc. degree in physics from RWTH Aachen University. Afterwards, he joined the Center for Computing and Communication there. He is responsible for the network infrastructure of the university and developed the first version of Blast-o-Mat.

*hektor@rz.rwth-aachen.de*

Thorsten Holz holds an M.Sc. degree in computer science from RWTH Aachen University and is currently a Ph.D. student at the University of Mannheim. His research focuses on honeypots and honeynets and currently one main area of work is botnets.

*thorsten.holz@informatik.uni-mannheim.de*

**AT RWTH AACHEN UNIVERSITY, WITH** about 40,000 computer-using people to support, we have built a system to detect infected machines based on honeypots. One important building block of Blast-o-Mat is Nepenthes, which we use both to detect malware-infected systems and to collect malware. Nepenthes is a low-interaction honeypot that appears as vulnerable software but instead decodes attack code and downloads malware. We have been successful at uncovering and quarantining infected systems with sensors listening at 0.1% of our address space. Investigation of collected malware has led to discovery of many infected systems and even a huge cache of stolen identity information.

The Internet has evolved into a platform for all kinds of security-sensitive services and applications. Online banking and payment have become part of today's way of life. For this reason, even home computers store valuable information such as passwords to online shops, credit card numbers, account data, and personal identification numbers. Therefore, securing network hosts, learning attack methods, capturing attack tools, and studying motives of computer criminals are important tasks for network administrators and security engineers.

One important aspect of network attacks is malicious software (*malware*) that spreads autonomously over the network by exploiting known or unknown vulnerabilities. In the form of *network worms* or *bots/botnets*—networks of compromised machines that can be remotely controlled by an attacker—malware poses a severe threat to today's Internet. For example, botnets cause damage from Distributed Denial-of-Service (DDoS) attacks, sending of spam, identity theft, and similar malicious activities.

Within the university network, we want to detect infected hosts as fast as possible. Only if we detect a compromised machine can we contain it and stop the spreading mechanism. This cessation protects other vulnerable hosts within the university network and also within external networks. Instead of using a classical Intrusion Detection System (IDS), we have built our own solution called Blast-o-Mat. This system aims at automatic notification and handling of malware-infected

hosts. The main task of Blast-o-Mat is to determine the person responsible for a system for which it receives an alert, send out a warning to the owner, and, if an infected host is still active after a certain period of time, block network access to and from this host. It is automatically transferred to a *quarantine network* (i.e., all access to the Internet is rerouted to a certain server). Basically, the infected machine can then only access certain sites to download patches and (in the future) also antivirus software. Within the quarantine network, we can also monitor what happens to the machine from a network point of view. As a result, we have a tool that automatically performs the time-consuming tasks that the network administrator normally has to carry out.

The system consists of several modules that try to detect an infected system:

- Blast-Sniffer continuously reads traffic data from a SPAN or mirror port of a central router of the network and writes it to a MySQL database. This database serves as the input for the next two intrusion detection sensors.
- Blast-PortScan detects hosts that are scanning a large number of IP addresses for certain ports, which could indicate a malware-contaminated machine. To accomplish this task, the module counts the number of TCP SYN packets sent by each host during a preconfigured period of time. Within our environment, a threshold of 50 SYN packets within three minutes has proven to be a reasonable indicator that tends not to generate false positives and is capable of detecting infected hosts efficiently.
- Blast-SpamDet aims at detecting machines that send spam messages. Similar to the portscan detector, it counts the number of initiated connections from a suspicious host, but this time only connections to mail servers are considered. When a certain number of connections are made, the server entity, with the help of packet-capture tools, starts to gather email header information of the suspected host. All used sender addresses are filtered and counted. If the number of unique sender addresses exceeds a certain threshold, further actions are initiated.
- Nepenthes is a low-interaction honeypot solution that is capable of automatically downloading malware. We describe its inner workings in a separate section. We use this module to get in-depth information about ongoing network attacks, and the analysis of the downloaded binary can help us to further examine the incident.

In the following, we give a brief background of honeypots and then introduce Nepenthes in detail. We show how this low-interaction honeypot can be used to detect infected machines, and we explain possible ways to isolate such compromised hosts. Finally, we highlight some incidents detected within the past couple of months.

## Background on Honeypots and Honeynets

A *honeypot* is "an information system resource whose value lies in unauthorized or illicit use of that resource" [1]. This methodology has been used to study attackers and types of attacks in depth, providing valuable information about tools, tactics, and motives of attackers. To learn more about malware, we use *low-interaction honeypots* such as Nepenthes [2] and *high-interaction honeypots* such as GenIII honeynets [3].

Low-interaction honeypots emulate services or operating systems. They allow an attacker limited interaction with the target system and allow us to learn mainly quantitative information about attacks. Since low-interaction

honeypots use simulation, they construct a controlled environment and thus the risk involved is limited. We give a more detailed introduction to low-interaction honeypots in the following section.

In contrast, high-interaction honeypots do not emulate any services, functionality, or operating systems. Instead, they provides real systems and services, allowing us to capture extensive information on threats. Several honeypots can be combined into a network, called a *honeynet*. We can capture the exploits of attackers as they gain unauthorized access, monitor their keystrokes, recover their tools, and learn what their motives are. The disadvantage to high-interaction solutions is that they have increased risk: Because the attackers can potentially fully access the operating system, they can potentially use it to harm other nonhoneypot systems.

A honeynet creates a fishbowl environment that allows attackers to interact with the system, while giving the operator the ability to capture all of their activity. This fishbowl also controls the attacker's actions, mitigating the risk of them doing harm to any nonhoneypot systems. The key element in a honeynet deployment is called the *Honeywall*, a layer-two bridging device that separates the honeynet from the rest of the network. This device mitigates risk through data control and captures data for analysis. Tools on the Honeywall allow for analysis of an attacker's activities. Any inbound or outbound traffic to the honeypots must pass through the Honeywall. Information is captured using a variety of methods, including passive network sniffers, IDS alerts, firewall logs, and the kernel module known as Sebek [4]. The attacker's activities are controlled at the network level, with all outbound connections filtered through both an intrusion prevention system and a connection limiter.

Neither of these two approaches is superior to the other; each has unique advantages and disadvantages.

## Collecting Malware with Nepenthes

The low-interaction honeypot Nepenthes aims at capturing malicious software such as network worms or bots that spread in an automated manner. The main focus of this application is to obtain the malware itself, i.e., to download and store the malware binary for further in-depth analysis. Unlike other low-interaction honeypots, Nepenthes does not emulate full services for an attacker to interact with. The key idea is to offer only as much interaction as is needed to exploit a vulnerability. For this reason, Nepenthes is not designed for any human interaction, as the trap would be easily detected. On the contrary, for the automated attack just a few general conditions have to be fulfilled, thus maximizing the effectiveness of this approach. These conditions usually include displaying the correct banner information of an emulated service and sending back specific information at certain offsets during the exploitation attempts. Therefore, the resulting service is only partially implemented. This allows deployment of several thousands of virtual honeypots with only moderate requirements in hardware and maintenance.

Nepenthes is designed as a single-threaded core daemon, with a number of different modules, facilitating each task of the malware collection process:

- *Vulnerability modules* emulate the vulnerable parts of network services.
- *Shellcode parsing modules* analyze the payload received by one of the vulnerability modules. These modules analyze the received shellcode, an assembly language program, and extract information about the propagating malware.

- *Fetch modules* use the information extracted by the shellcode parsing modules to download the malware from a remote location.
- *Submission modules* take care of the downloaded malware (e.g., by saving the binary to a hard disk, storing it in a database, or sending it to antivirus vendors).
- *Logging modules* log information about the emulation process and help get an overview of patterns in the collected data.

Besides the modular structure, Nepenthes provides an event-driven notification mechanism. Each step of an attack triggers certain events, which other modules can register and therefore react on. As a result, Nepenthes can be highly customized to fit into new environments.

We briefly describe the three most important kinds of modules in more detail in order to give a better understanding of the operation of Nepenthes.

Vulnerability modules are the main reason for the efficiency of the Nepenthes platform. The main idea—only emulating the vulnerable parts of a service—has already been explained. We only need to emulate the relevant parts and are thus able to efficiently implement this emulation. Eventually, we receive the actual payload, which is then passed to the next type of module.

Shellcode parsing modules analyze the received payload and automatically extract relevant information about the exploitation attempt. Currently, only one shellcode parsing module is capable of analyzing all shellcodes received in the wild. The module works in the following way: First, it tries to decode the shellcode. Most shellcode is obfuscated with an XOR encoder. An XOR decoder is a common way to "encrypt" the native shellcode in order to evade intrusion detection systems and string-processing functions. After decoding the code itself according to the computed key, this module then extracts more information from the shellcode (e.g., credentials). If enough information can be reconstructed to download the malware from a remote location, this information is passed to the next type of module.

Fetch modules have the task of downloading files from remote locations. Currently, there are several different fetch modules. The standard protocols TFTP, HTTP, and FTP are supported. Since some bots use custom protocols for propagation, there are also fetch modules to handle these bot-specific protocols.

The modularity and flexibility of Nepenthes allow for the deployment of unique features not available in high-interaction honeypots. For example, it is possible to emulate the vulnerabilities of different operating systems and computer architectures on a single machine during a single attack (e.g., an emulation can mimic the generic parts of a network conversation and, depending on the network traffic, decide whether it needs to be a Linux or a Win32 machine).

The source code of Nepenthes is available under GPL at http://nepenthes.mwcollect.org. In addition, a more detailed introduction, together with preliminary results, is also available [2].

## NEPENTHES AS PART OF AN IDS

Within the Blast-o-Mat architecture, Nepenthes serves as a sensor to detect infected machines. These machines typically try to propagate further by scanning for vulnerable machines. Thus we have placed Nepenthes sensors all over the network and on each of these IP addresses they emulate common vulnerabilities, as already explained. We use about 180 IP addresses to cover three /16 networks, thus covering about 0.1% of all addresses. Nevertheless, preliminary results show the effectiveness of this approach. One important finding is that Nepenthes has not generated any false positives: Whenever Nepenthes signals a successful exploitation attempt, it is not a portscan or misconfigured system, but a real intrusion attempt. To this point, the Blast-o-Mat system has already detected hundreds of infected machines and the automatic containment works without problems.

## MITIGATION OF INFECTED SYSTEMS

As soon as an infected system has been detected, the first question entails how to deal with it. Presumably the best way is to immediately take the system offline, giving it as little chance as possible to infect other systems in the network. The inhibition can take place on any of the OSI layers, depending on the given infrastructure. If direct access to the switch port of the conspicuous machine is given, one can disable this port. In this case the host is locked at the physical layer of the OSI model. An inhibition on layer 2 is equal to blocking the MAC address of the hostile host. This approach would also prevent the system from being taken online again on a different switch port. The disadvantage of these two methods is the effort it takes to determine to what network device the contaminated machine is connected. Less costly is the locking of the IP address with the help of access lists (OSI layer 3). In this case, we need to determine the router, which routes the appropriate network. Although the host is properly blocked, it can still infect systems within the same local area network (LAN). On higher layers of the OSI model, it is possible to lock certain TCP or UDP ports or operate different protocol-specific filters to isolate an infected host. However, all modifications to network components have to be reverted as soon as the problem is solved and the user wants to get back online.

A different approach to taking a contaminated host offline is to place it into a *quarantine network*, isolating it from other systems. Although this requires a certain infrastructure, this is the most effective solution, as additional information can be collected from the quarantined host. Currently, we have implemented a simple form of such a quarantine network: The Blast-o-Mat is capable of redirecting HTTP traffic of infected machines to a special Web server. Before taking a look at the practical implementation of this approach, we introduce two ways to actually block the infected host.

When blocking, we differentiate between two groups of users: static IPs (normally staff people or PC pools) and dynamic IPs (typically WLAN users).

To identify the responsible person(s) for hosts with static IP addresses, we maintain an XML-based database with all relevant information. For each subnet the database contains the registered administrators, their phone number and email address, the net mask, the acronym of the institute, and, if available, the assigned Virtual Local Area Network (VLAN) number. Additionally, for each entry there exists information about the manageable network router through which the associated subnet is routed. To lock a

host with a static IP address, use is made of a Perl script capable of automatically creating antispoofing access lists. These access lists can be extended with firewall rules or, in our case, with lists of locked machines, thus efficiently blocking contaminated hosts from accessing the network.

To identify the person responsible for a dynamically assigned IP address, we have to ascertain the account name from the authentication or accounting server. Therefore, we have to compare the IP address and the time of the incident with the information stored in the Radius server. We run a slightly modified version of the FreeRadius software, which writes its accounting data to a MySQL database, on a daily basis. Thus, we have a database table for each day, which greatly accelerates the process of searching for specific accounting data. The account locking of an infected host is accomplished by setting a special flag in the LDAP database, which is used for user authentication. Once the flag is set, a user with an infected machine can no longer connect to the campus network and has to contact the helpdesk to be unlocked again.

One of the more complicated tasks in automatic locking of infected systems is to notify the user of the suspected host. Our main method is to notify any responsible person via email. Since every student at RWTH Aachen gets his or her own email address upon enrollment, we have a fairly good possibility of reaching any student. The obvious limitation is that we cannot  besure that the students read their university email frequently or even at all. Therefore the Blast-o-Mat is capable of redirecting certain traffic to a specially designed Web server (as briefly mentioned above). Because of the network structure at RWTH Aachen, the redirection currently works only for the wireless network, but we hope to extend this in the future. All traffic of wireless hosts has to pass one central gateway. Thus, we are able to efficiently redirect any traffic of hostile hosts at this point, via the use of certain iptables rules. The main advantage of this approach is that the responsible person of a redirected host is efficiently informed, even if the warning mail of the Blast-o-Mat is not read. Every attempt to open a Web site on a redirected host displays the information site of the quarantine Web server, showing all gathered data about the incident so far. Furthermore, email delivery is still possible, allowing the user to get additional information, provided in the Blast-o-Mat warning messages.

To achieve the redirection of a contaminated host, the Blast-o-Mat remotely executes a Python script on the gateway server and transmits the account name, the IP address, and the time the system was online as parameters. The easiest way would be to do the redirection based on the IP address of the infected host. But since we have to deal with dynamically assigned addresses, this would not prevent the user from logging in again with a different IP address, thus circumventing the redirection measures. Therefore, we have to determine the MAC address of the offending machine. This is accomplished with the help of an additional script which queries the DHCP server with the time the host was online and its IP address as parameters. Every DHCP server maintains a lease file, containing all MAC addresses of hosts to which it assigned an IP address, together with the time interval the given IP address is valid. With the help of this file, we are able to determine the MAC address of the system that was online with a certain IP during a given time. As a result, the script generates an iptables rule that redirects any further HTTP traffic of the specified MAC address to the quarantine Web server.

A more advanced solution to building a quarantine network would involve VLANs: As soon as a host is detected by the Blast-o-Mat, the VLAN tag for

this machine is changed to the tag of the quarantine network (which could be a honeynet). As a result, all traffic is redirected. The major drawback of this concept is that it requires the network infrastructure to allow access to the switch port of each host; additionally, the switch must support VLAN tagging of certain ports.

In the next section, we take a closer look at one particular alert generated by Nepenthes.

## A Modern Trojan: Haxdoor

During one security incident detected by Nepenthes in April 2006 we noticed a strange behavior of the infected machine: It constantly tried to post data to a certain PHP file located on a server in the United States. Since the machine had already been moved into the quarantine network, we could observe it further. We noticed that sensitive data—in this case passwords—was sent to the remote server. A closer examination revealed the URL from the HTTP requests and we quickly noticed that these requests were caused by a variant of Haxdoor, one of the most advanced Trojans in the wild.

In addition to the normal Trojan capabilities, such as copying itself to the Windows Installation Directory and start on reboot, Haxdoor also implements rootkit capabilities and advanced identity theft mechanisms. It can, for example, hide its presence on the compromised machine via SSDT (System Service Dispatch Table) hooking, as well as steal all information entered into Internet Explorer. All of this captured information can be sent to a central server, which is precisely the activity we observed within the quarantine network.

During further investigation, we found several log files that contained all information stolen from all infected machines. In total, these log files contained more than 6.6 million entries, amounting to 285 MB of data. This data was stolen from the compromised machines between April 19 and April 27, 2006, i.e., within only nine days. In total, we found evidence of more than 39,000 IP addresses that were victim of this particular Haxdoor infection. These numbers show the effectiveness of this kind of attack.

The log files contained full detailed information about more than 280 bank accounts and several credit card numbers. All major German banks were victim of this incident and several large brands from the e-commerce sector were also targeted. In addition, the attacker also collected sensitive information such as username and password combinations and other data entered into HTML forms from the victims' computers. More information about this kind of modern data theft can be found in the article by Team CYMRU in this issue of *;login:* [5].

We handed this information over to DFN-CERT, the Computer Emergency Response Team responsible for German research and education networks. The affected users were warned, as were universities, ISPs, and other affected sites.

### BINARY ANALYSIS

Sandboxing is a well-established approach that involves executing the malware in an emulated environment and monitoring its behavior. During preparation of a diploma thesis at our lab, Carsten Willems developed a sandbox named CWSandbox [6]. Preliminary results show that CWSandbox is able to efficiently and accurately analyze a given malware binary.

The tool is able to extract all important information from a given binary in an automated way within a short amount of time (usually three minutes). The extracted information includes information about changes to the filesystem or the Windows registry, process access, DLL handling, and network communication. The whole analysis process does not require any human interaction and can be parallelized, allowing for concurrent analysis of large amounts of data.

The information in the following paragraphs is based on the reports generated by CWSandbox, enriched with information retrieved via manual binary analysis. We analyzed eight different variants of Haxdoor. All of them share many characteristics. The following description is a generalization of the different Haxdoor variants.

Typically, this specimen of malware creates several different files in the Windows installation folder. By default, this is either C:\Windows (Windows 2000 and XP) or C:\Winnt (Windows NT). The created files normally include two Dynamic Linked Libraries (DLLs), three to four drivers (SYS), and several additional configuration files. For example, according to Bitdefender, the variant Haxdoor.IN creates the following files: sndu32.dll and qm.dll (same as sndu32.dll), sndu64.sys and qm.sys, and stt82.ini, klgcptini.dat, and stt82.ini.

Upon executing, the binary loads several DLLs. These include the typical Windows DLLs such as kernel32 or ntdll but also network-related DLLs such as wsock32 and the code within the newly created files. One characteristic sign of Haxdoor is the creation of a mutex with the name RasPbFile.

Haxdoor also interacts with the Windows registry to enable a mechanism to be started upon reboot. In contrast to other malware, which commonly adds a registry key under Run or RunService, Haxdoor is more advanced. It uses a mechanism to auto-load via Winlogon or even during SafeBoot. The corresponding registry keys are:

HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify

HKLM\SYSTEM\CurrentControlSet\Control\SafeBootMinimal

HKLM\SYSTEM\CurrentControlSet\Control\SafeBootNetwork

Via the Windows Service Control Manager (SCM), Haxdoor also adds a service to the infected system that is automatically started upon system startup. The name of the service varies; it can, for example, be "SoundDriver SDB64" or "UDP32 netbios mapping," depending on the variant. In addition, it creates a remote thread within the memory space of Explorer.exe, in order to add some further services. Moreover, Haxdoor has some advanced tricks to hide its presence on the infected system, and it is hard to get rid of it. This topic is, however, beyond the scope of this article; more information can be found on the Web sites of antivirus vendors.

## Conclusions and Future Research

The Blast-o-Mat IDS had been running for about seven months as of September 2006, efficiently handling malware-infected hosts in the campus network of RWTH Aachen. With the help of the honeypot Nepenthes and the additional intrusion sensors, so far a total of 361 incidents have been detected. A little more than one-third were reported by Nepenthes, with the rest split between the Blast-PortScan and Blast-SpamDet sensors. The PortScan sensor reported the most incidents, owing to its much larger

number of monitored ports than vulnerability modules. However, each portscan that was detected on a port for which a vulnerability module exists was detected by Nepenthes as well.

Although its missing vulnerability modules means that Nepenthes does not recognize exploit attempts on all ports, it has proven to be a great intrusion detection mechanism. The biggest advantage is its accuracy, as no false positives are reported, as well as the high detection ratio, with only a few IP addresses assigned. Currently, we are monitoring with Nepenthes less than 0.1% of the complete IP space and already achieve almost the same results as the Blast-PortScan sensor, which receives its data from a SPAN port of a centralized router.

Because the bandwidth of current large-scale networks such as the one of RWTH Aachen already exceeds 1 gigabit of traffic and can approach 10 gigabits, common SPAN port monitoring will no longer work without the use of specialized and expensive hardware. However, Nepenthes will still deliver the same quantitative results with just 180 IP addresses. Therefore, it serves as a future-proof intrusion detection sensor, capable of running on a normal off-the-shelf computer.

In addition to the detection of contaminated hosts, Nepenthes also captures the malware that is trying to exploit the emulated vulnerabilities. Thus, we are able to submit the collected binaries for further analysis to different applications, such as virus scanners, to determine the kind of malware, or to the CWSandbox, to find out more about the behavior of malicious software. As a result, we are able to supply a qualitative high-class report for the detected incidents, both to help clean infected machines and to raise the user's security awareness.

## ACKNOWLEDGMENTS

We would like to thank Sam Stover for reading a previous version of this paper and giving valuable feedback that substantially improved its presentation. In addition, we would like to thank the Nepenthes Development Team for implementing such a wonderful tool.

## REFERENCES

[1] The Honeynet Project. Know Your Enemy: Honeynets (May 2005): http://www.honeynet.org/papers/honeynet/.

[2] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling, "The Nepenthes Platform: An Efficient Approach to Collect Malware," *9th International Symposium on Recent Advances in Intrusion Detection, RAID06, Hamburg, Germany, September 20-22, 2006, Proceedings, Lecture Notes in Computer Science* (Springer, 2006).

[3]E. Balas and C. Viecco, "Towards a Third Generation Data Capture Architecture for Honeynets," *Proceedings of the 6th IEEE Information Assurance Workshop, West Point* (IEEE, 2005).

[4] The Honeynet Project. Know Your Enemy: Sebek (November 2003): http://www.honeynet.org/papers/sebek.pdf.

[5] Team Cymru, "The Underground Economy: Priceless," *;login:,* this issue.

[6] CWSandbox—behavior-based binary analysis: http://www.cwsandbox.org/.

See also T. Holz, "Spying with Bots," *;login:*, 30 (6) (Berkeley, CA, 2005): 18–23.

ANDY OZMENT AND
STUART E. SCHECHTER

# the security of OpenBSD

## MILK OR WINE?

Andy Ozment is a Ph.D. student in the Computer Security Group at the University of Cambridge. He will graduate in July 2007. This article describes work performed in part when Andy was on the technical staff at MIT Lincoln Laboratory.

*andy.ozment@ieee.org*

Stuart E. Schechter is a researcher in computer security at MIT Lincoln Laboratory. He explores security problems as they relate to system design, economics, and user interfaces. Ironically, Stuart can neither digest milk nor tolerate the taste of wine.

*ses@ll.mit.edu*

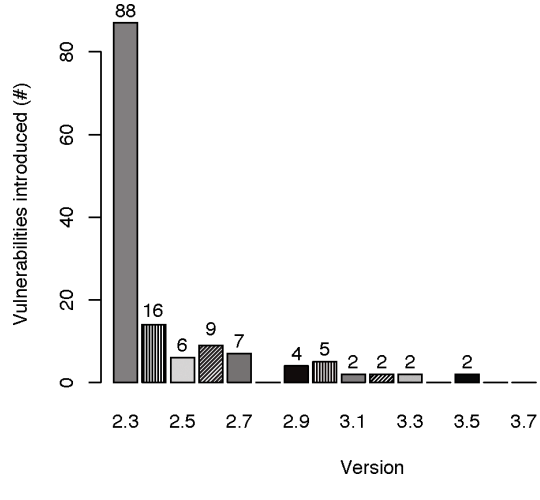



**FIGURE 1: THE NUMBER OF VULNERABILITIES INTRODUCED IN EACH VERSION RELEASED DURING THE STUDY**

**PURCHASE A FINE WINE, PLACE IT IN** a cellar, and wait a few years: The aging will have resulted in a delightful beverage, a product far better than the original. Purchase a gallon of milk, place it in a cellar, and wait a few years. You will be sorry. We know how the passing of time affects milk and wine, but how does aging affect the security of software?

Many in the security research community have criticized software developers both for releasing software with so many vulnerabilities and for the lack of any apparent improvement in this software over time. However, critics have lacked quantitative evidence that applying effort over time will result in software with fewer vulnerabilities. In short, we don't know whether software security is destined to age like milk or has the potential to become wine.

We thus investigated whether or not the rate at which vulnerabilites are reported in OpenBSD is decreasing over time. For a more technical description of this work, see [1].

## Vulnerability Data

We compiled a database of 140 vulnerabilities reported in the 7.5 years between 19 May 1998 and 17 November 2005. Vulnerabilities were identified by merging data from the OpenBSD security Web page and four public vulnerability databases: NVD (formerly ICAT), Bugtraq, OSVDB, and ISS X-Force.

Figure 1 shows the number of vulnerabilities that were "introduced" in each of the fifteen versions of OpenBSD that were released during our study. A vulnerability is counted as having been introduced in a version if that version is the first to contain the vulnerability within its source code.

How do we know when a vulnerability was introduced? Vulnerability or patch reports often list the versions affected by that vulnerability; however, vendors and vulnerability hunters rarely bother to test more than two or three versions back. So vulnerability and patch reports are not a sufficiently accurate means of finding the release in which a vulnerability was introduced.

Instead, when vulnerabilities were reported, we used the patch to identify the vulnerable code in the OpenBSD CVS repository. We then tracked that code back through all the previous versions

of OpenBSD until we could identify when it had first been checked into the code base. The first release that included that vulnerable code is the release to which the vulnerability is attributed.

Our analysis covers all portions of the OpenBSD code in the OpenBSD team's primary CVS repository. This includes the X-windowing system, the Apache Web server, and many additional services not traditionally considered to be part of the core operating system. However, it excludes the "ports" collection of third-party software, which is not officially part of OpenBSD. We started our study with version 2.3, which we refer to here as the "foundation version," because it was the first source-code stable release for which all reported vulnerabilities are documented.

During the study, versions of OpenBSD were released approximately every six months. The vulnerabilities that were introduced in each version were usually checked into the CVS repository during the six months prior to that version's release. For example, the vulnerabilities attributed to version 2.4 were introduced in the six months between its release and the release of the prior version. The one exception to this rule is the foundation version (2.3): all vulnerabilities introduced before this version's release are attributed to this version. This includes more than twenty-five years since coding on Berkeley UNIX began. As a result, we see in Figure 1 that 62% of the vulnerabilities reported during the study were introduced in the foundation version.

## Source Code Evolution

The majority of vulnerabilities reported during the study were thus introduced sometime prior to the release of the foundation version. But now, 7.5 years later, does the security of the foundation version have any relevance to current versions of OpenBSD?

To answer this question, we investigated the proportion of the source code in the most recent version of OpenBSD that remains unchanged since each earlier version. Figure 2 shows the results of our analysis. Each column represents a composite version; each row represents a source version that contributes code to the composite. A line of code in a composite version of OpenBSD is said to originate in a source version if the line was last modified in that source version. The column for version 2.3 is composed of a single row: By definition, all code in this foundation version is said to originate in it. For each successive version, a new row is added to the column to represent the lines of code that were altered or introduced in that release.

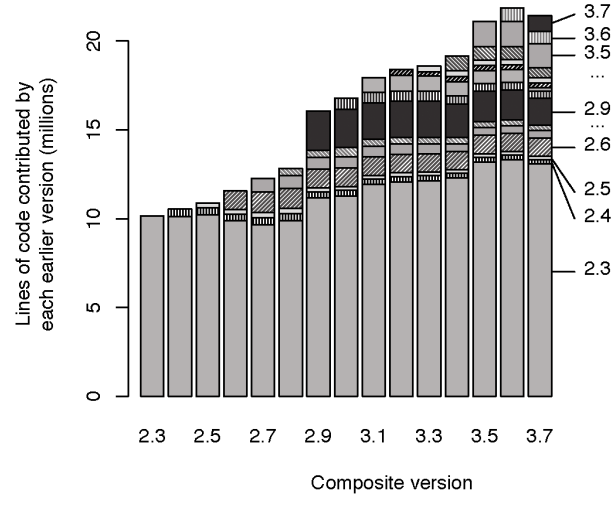



FIGURE 2: THE COMPOSITION OF THE FULL SOURCE CODE

Identifying how the source code evolved over time was a difficult project. We first preprocessed each version of the source code. Only files with the suffix .c or .h were retained, and all comments were stripped. We then compared each version with each successive version. We used diff to compare files with the same path and filename. The diff tool was instructed to ignore changes in whitespace and the location of line breaks. Finally, we counted the number of lines in each version that were unchanged from the immediately prior version. By recursively repeating this process, we obtained the data in Figure 2.

The resulting estimate of code commonality is highly conservative. The diff tool marked code lines as changed even for trivial alterations such as variable renaming and some types of reformatting—and the OpenBSD team has been reformatting the code base. In addition, if a file from a previous version was moved or copied to a new location and if even one line of the file in the new location was changed, our analysis will treat the entire file as new. Furthermore, if the name of a file is changed, then all of the code in that file is treated as new. Our comparison results will thus understate the degree to which later releases are composed of  that is substantively unchanged from earlier releases.

Despite our conservative methodology, Figure 2 shows that unchanged code from the foundation version still comprises 61% of the code in version 3.7—which was released over seven years later. The security of the source code for the foundation version is thus still pertinent to the security of the source code in current versions of OpenBSD.

However, there is another startling result that is visible in Figure 2. The number of lines of source code contributed by the foundation version to each composite version changes over time. That, in itself, is unsurprising. What is surprising is that the number of lines *increases*. How is it that the foundation version contributes more lines of code to version 3.7 than were in the foundation version itself? We discovered that the lines of code derived from the foundation version increases over time because developers reused source code files in different locations. For example, one copy of a compression library file may be used to generate a shared library while another copy of the same file may be used to compile the kernel. Code recycling via source-file replication causes a net increase in the lines of code that are present in later versions.

Several large alterations and/or introductions of code stand out in Figure 2: versions 2.6, 2.9, and 3.5. The magnitude of the changes in versions 2.6 and 3.5 is primarily due to a large number of files being renamed and slightly altered. Our current methodology thus overstates the number of new lines of code and understates the contribution of code derived from earlier versions. The changes in version 2.9 are caused in part by the renaming of files; however, they were also the result of a major upgrade of the XFree86 package.

## Milk or Wine?

Let's return to our original question: Does software security improve with age? Unfortunately, we don't have enough vulnerability reports to analyze most versions of OpenBSD. Only the foundation version provides us with enough information: 87 "foundational vulnerabilities" were reported. Our question then becomes: Is the rate of vulnerability reporting for OpenBSD version 2.3 decreasing?

We use three different approaches to answering this question. First, we divide the study into two halves and count the number of vulnerabilities reported in each half. Figure 3 shows the result, with confidence intervals calculated by assuming that vulnerability reporting is a homogeneous Poisson process. The number of vulnerabilities reported significantly declines from the first half (58 vulnerabilities) to the second (28 vulnerabilities).

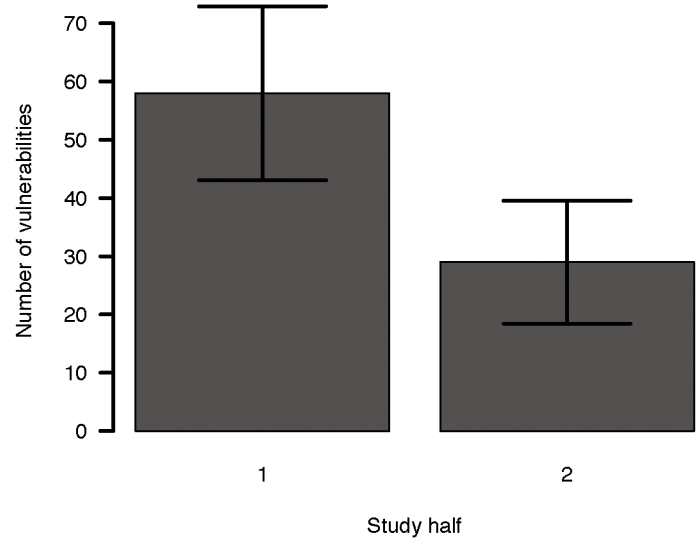



**FIGURE 3: THE NUMBER OF FOUNDATIONAL VULNERABILITIES REPORTED DURING EACH HALF OF THE STUDY**

The next approach is to utilize a Laplace test, in which the discovery of vulnerabilities is assumed to be a heterogeneous Poisson process. The test assesses whether the interarrival times of vulnerability reports are decreasing. We use as our data the number of days elapsed from the identification of one foundational vulnerability to the next.

When the calculated Laplace factors are less than the lowest horizontal dotted line in Figure 4, the data indicates a decreasing rate of vulnerability reporting, with a two-tailed confidence level of 95%. The test finds evidence for a decrease in the rate of vulnerability reporting by the end of year four; by year six, the evidence for a decrease in the reporting rate is statistically significant. This test therefore also supports the conclusion that the rate at which foundational vulnerabilities are reported is declining.

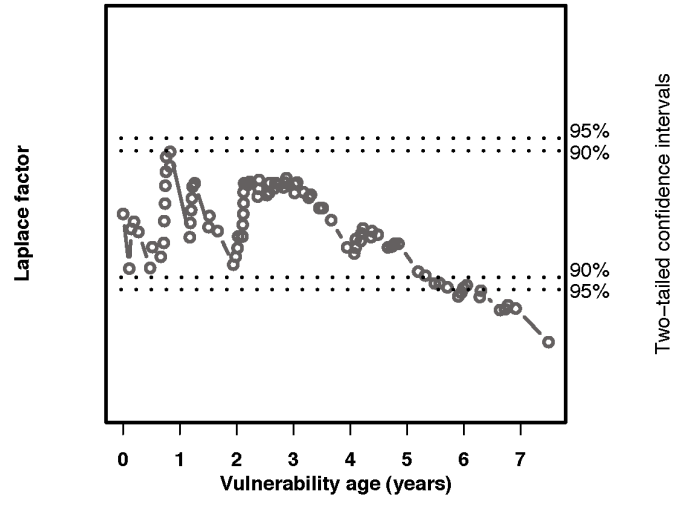



**FIGURE 4: LAPLACE TEST FOR THE EXISTENCE AND DIRECTION OF A TREND IN THE RATE OF VULNERABILITY REPORTING**

In our third approach, we attempt to fit reliability growth models to our data. Although normally applied to the more random discovery of defects, these models can also be applied to the reporting of vulnerabilities. We analyzed the data with seven time-between-failures reliability growth models. One of the seven models had acceptable one-step-ahead predictive accuracy and goodness of fit for the data set: Musa's logarithmic model. According to this model, the number of vulnerabilities expected to be reported on a given day decreases from 0.051 to 0.024 over the course of the study. Furthermore, it estimates that 67.6% of the vulnerabilities in the OpenBSD 2.3 source code have now been found.

Each of our three approaches thus indicates that the rate of foundational vulnerabilities reported is decreasing.

### CAVEATS

The rate at which vulnerabilities are discovered and reported depends on the level of effort being expended by vulnerability hunters. To measure how much more difficult it has become to find vulnerabilities over time, we would need to normalize the rate of discovery by the effort being exerted and the skills of those exerting it. Unfortunately, vulnerability reports do not include estimates of how many individuals were involved in examining the software, the time they spent, or their relative skills. Our analysis thus can only show that, in the vulnerability hunting environment that existed during our study, the rate of vulnerability reporting decreased for the foundation version.

## While We're at It

Our data prompted us to consider two other questions:

- What is the median lifetime of a vulnerability?
- Do larger code changes have more vulnerabilities?

To answer the first question, we calculate the median lifetime of reported vulnerabilities for the foundation version. The median lifetime is the time elapsed between the release of that version and the death of half of the vulnerabilities reported in that version. Alas, we don't know how many vulnerabilities remain in the foundation version or when they will be found. As a result, we can provide only a lower bound on the median lifetime. The result is striking: It took at least 2.6 years to find half of all the foundational vulnerabilities that would be found during the 7.5-year study period.

The second question is related to "vulnerability densities." Software engineers have examined the defect density of code: the ratio of the number of defects in a program to the number of lines of code. Some have argued that any well-written code can be expected to have a defect density that falls within a certain range (e.g., 3–6 defects per thousand lines of code). Our second question is thus whether or not there is a linear relationship between the number of lines of code altered and/or introduced in a version of OpenBSD and the number of vulnerabilities introduced in that version.

As we cannot measure the total number of vulnerabilities present, we measure the number discovered within four years of release for each version that is at least four years old. Figure 5 illustrates the relationship between the number of lines of altered and/or introduced code and the number of vulnerabilities reported. Neither a visual examination of the fi-

gure nor Spearman's rho test finds a correlation between the number of lines of code altered and/or introduced in a version and the number of vulnerabilities introduced.

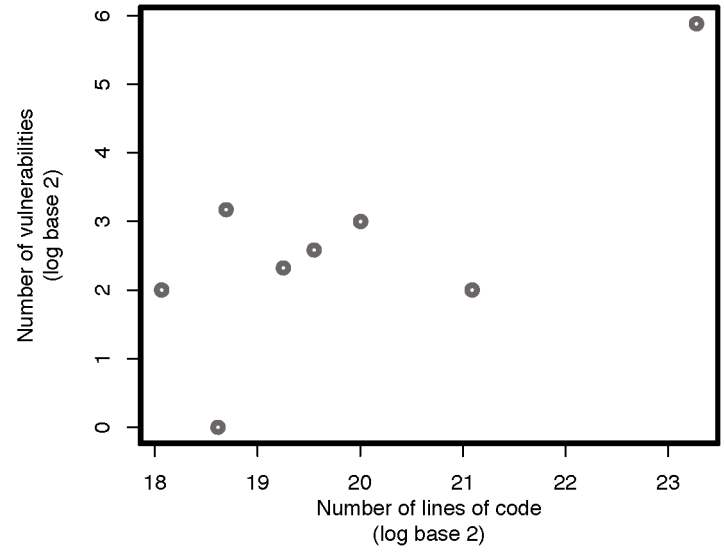



**FIGURE 5: THE NUMBER OF VULNERABILITIES INTRODUCED AND REPORTED WITHIN FOUR YEARS OF RELEASE COMPARED TO THE NUMBER OF LINES OF CODE ALTERED OR INTRODUCED, BY VERSION**

When calculated per thousand lines of code, the density of all *reported* vulnerabilities ranged from 0 to 0.033 and averaged 0.00657. There appears to be no trend with respect to the densities increasing or decreasing in newer code. These vulnerability densities are thus three orders of magnitude less than the normal range of defect densities. However, the two figures are not necessarily contradictory: Defects include both vulnerabilities and bugs that are not vulnerabilities. Furthermore, multiple identical security defects that were discovered at the same time are considered a single vulnerability in our data.

## Conclusion

Over a period of 7.5 years and 15 releases, 62% of the 140 vulnerabilities reported in OpenBSD were foundational, that is, they were present in the code at the beginning of the study. It took more than 2.6 years for the first half of these foundational vulnerabilities to be reported.

We found that 61% of the source code in the final version studied is foundational: It remains unaltered from the initial version released 7.5 years earlier. The rate of reporting of foundational vulnerabilities in OpenBSD is thus likely to continue to greatly influence the overall rate of vulnerability reporting.

We also found statistically significant evidence that the rate of foundational vulnerability reports decreased during the study period. We utilized a reliability growth model to estimate that 67.6% of the vulnerabilities in the foundation version had been found. The model's estimate of the expected number of foundational vulnerabilities reported per day decreased from 0.051 at the start of the study to 0.024. We thus conclude that the foundation version of OpenBSD is like wine: It is growing more secure with age.

## DISCLAIMER

This work is sponsored by the I3P under Air Force Contract FA8721-05-0002. Opinions, interpretations, conclusions, and recommendations are those of the author(s) and are not necessarily endorsed by the United States Government.

This work was produced under the auspices of the Institute for Information Infrastructure Protection (I3P) research program. The I3P is managed by Dartmouth College and supported under Award number 2003-TK-TX-0003 from the U.S. Department of Homeland Security, Science and Technology Directorate. Points of view in this document are those of the authors and do not necessarily represent the official position of the U.S. Department of Homeland Security, the Science and Technology Directorate, the I3P, or Dartmouth College.

## REFERENCE

[1] A. Ozment and S.E. Schechter, "Milk or Wine: Does Software Security Improve with Age?" *Proceedings of the 15th USENIX Security Symposium* (Berkeley, CA: USENIX Association, 2006).

NICHOLAS WEAVER AND DAN ELLIS

# white worms don't work

Nicholas Weaver is a researcher at ICSI specializing in worms, computer security, and architectures for high-speed intrusion detection.

*nweaver@icsi.berkeley.edu*

Dan Ellis recently finished his Ph.D. at George Mason University. At MITRE, as an infosec scientist, he's been working on various aspects of worm technology since 2001.

*ellisd@mitre.org*

**SEVERAL VOICES HAVE REPEATEDLY** proposed using worms as a tool for fighting other worms, updating systems, and other security tasks. Such *white worms* (also known as anti-worms [2,5,17], predators [6], or nematodes [1]) have been proposed as a mechanism to counter a spreading worm. The idea is to launch a worm that spreads to patch the target and make it immune to the competing worm. Likewise, there have been several allegedly white worms both written (e.g., Code Green [7]) and released (e.g., Welchia [16] and Anti-Santy Worm [8]) with the alleged goal of cleaning up an existing infection.

Using white worms to immunize from or clean up after a competing worm is a bad idea for several reasons. Technically, writing worms to do what is desired is incredibly difficult. It is difficult to target a white worm to infect and immunize all of the machines of interest and *only* those machines. Also, getting the worm payload to do exactly what is desired is hard to do with significant testing, let alone on the fly in response to another worm. Consequently, the risk of the worst-case, compounded problem (deploying a damaging payload across a large number of hosts that do not belong to you) is practically impossible to mitigate. Even using a highly controlled worm for penetration testing on a live network [1] has serious issues.

There are also some significant legal reasons not to use a white worm. For example, infecting a machine that one does not own is a serious crime in most nations, independent of the intentions of the author of the white worm. Worse, if there is a problem in the targeting of the worm and the payload of the worm, significant damage is possible, which could even lead to international conflict if spread occurs across national boundaries, independent of intent or sponsorship [18].

Fortunately, a combination of machine attestation, automated inventory management, and patch management offers a superior and acceptable solution. A combination of these tools provides mechanisms to distribute patches and protective instructions faster than all existing worms. Even better, the fundamental time scale for patch distribution can equal or even exceed the fastest worm theoretically possible. Any host that could be

configured in any way to improve or facilitate the performance of a white worm could also be instrumented with a patch-management system. We conclude that, without exception, the use of white worms poses an unacceptable risk.

## What Is a White Worm?

A *white worm*, in our definition, is any worm released with allegedly benign intent. The objective could be to patch systems before a malicious worm is released, to out-compete an already spreading worm, or to clean up after a worm attack. We use the term *white* rather than *good* because the actions of such a worm may not be deemed beneficial by the owner of a targeted machine.

In all these scenarios, we assume that the white worm is spreading by exploiting some vulnerability in a target system, rather than by legitimately accessing the target machine (e.g., via an authorization daemon installed on the target). This is simply because if an authorized backdoor or daemon can be installed on the target computers, then a patch-management system can be deployed instead.

A patch-management system, in contrast to a white worm, is a purely consensual mechanism. All participating systems contain a small daemon that is used to receive and apply defensive instructions, including patches and network filters. Combined with machine attestation systems such as Cisco's Network Admission Control [3], this system can ensure that all systems within a well-structured corporate network are participating and able to receive and process updates.

## Displacing an Existing Worm

The most commonly proposed application for a white worm is to displace a malicious worm already in place. Yet it is currently considered standard procedure for an attacker to close the vulnerability used to exploit the system, and we have heard anecdotal reports of even other unrelated vulnerabilities being patched. Compromised hosts are a resource to the attacker; thus, an attacker benefits from preventing that resource from being claimed by others.

As a specific example, Welchia [16] was called a white worm, because it removed a competing worm (Blaster) and installed the patch. Yet, Welchia also contained a malicious payload, opening up a backdoor on infected systems. Welchia's patching was really to prevent double-infection and to remove a competitor that made infected systems unstable.

Displacing a rootkit or worm that takes measures to defend itself can prove highly difficult. Since a worm author is likely to be less concerned about the stability of other applications on the target host, there is little motivation for a defensive worm author not to produce a patch suite. Whereas a conscientious sysadmin would vet (which may include testing negative impacts) a patch before applying it, a worm author has little motivation. Therefore a worm author may decide to patch other vulnerabilities not previously addressed by the sysadmin. As a white worm would be constructed primarily using publicly available exploits, it would be very difficult to successfully infect a previously infected and patched host.

A worm author may get the benefit of limiting access without patching if infection results in the host becoming invulnerable, which often occurs

when the target application is single-threaded and the exploit never returns control flow to the victim program. Blaster on Windows 2000 [15], Witty [11], and Slammer [10] all showed this behavior. Additionally, the worm author could simply terminate the vulnerable service after infection, unless the worm author needs to use the vulnerable service or the service is essential to system operation.

Thus, unless there is a *separate* vulnerability known by the white-worm author but not by the original worm author, there is a vulnerability in the original worm, or the original worm author simply doesn't care about being displaced, a white worm can't be reliably used to displace an existing infestation.

Of course, the same problem applies to patch management. It is common for malicious code to disable antivirus and other defensive applications in the process of rootkitting a system. Thus, although a white worm cannot reliably displace a worm, neither can patch management. But this is part of the general problem of recovering compromised machines, not a specific limitation of patch management.

## Outracing a Spreading Worm

Thus if a white worm is to out-compete a worm that is currently spreading, it must be created dynamically and released very quickly in response [2]. The problem is that if both worms are equally optimized, the worm with the head start has an effectively insurmountable advantage. Only if the anti-worm gets an unmatched performance boost through some other technique (e.g., hitlisting) can it be expected to compete with the initial worm.

However, the author of a malicious worm can use these same techniques to gain a speed advantage, obviating any optimizing effect of the white worm. Thus, it is best to consider the two worms, the initial malicious worm and the white worm, as having the same speed, as any speed improvements that the white-worm author could use could be copied by the author of the malicious worm.

We conducted a simple simulation using a modified version of the simulator from [14]. At $T = 0$, a malicious worm is released (300,000 vulnerable systems, 200 scans/s/worm, 32-bit IPv4 address space). After some specified fraction of the hosts are infected (0.01%, 0.1%, 1%, and 10%), an equivalent white worm is released from a single source and spreads at the same rate as the initial worm. For all but the 0.01% sensitivity, the white worm was effectively useless, having a negligible impact on the black worm's propagation and final infection. But even the supersensitive white worm, released when 0.01% of the systems are infected by the black worm, is almost useless, inoculating less than 3% of the victims in the median case. We conducted 100 simulations and graphed the median case, the case representing the top 5%, and the case representing the bottom 5% in terms of effectiveness. This is shown in Figure 1.

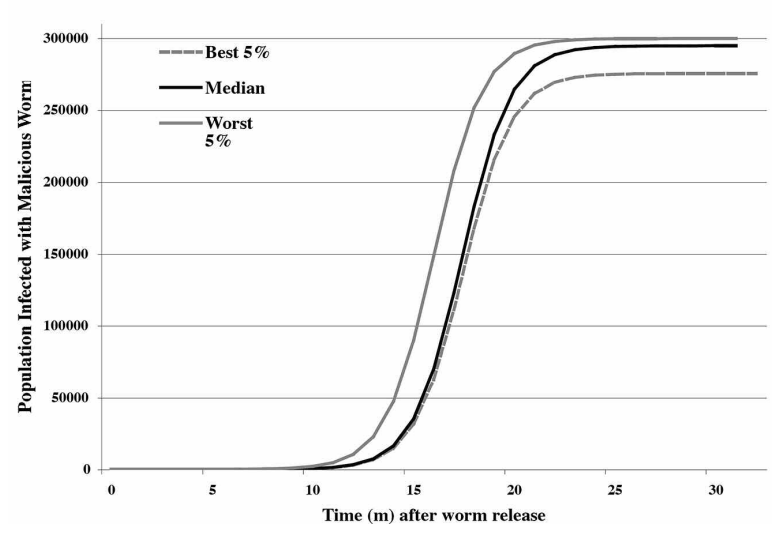



**FIGURE 1: THE NUMBER OF SYSTEMS INFECTED BY THE BLACK WORM FOR THREE DIFFERENT SIMULATION RUNS (OUT OF 100), WHEN THE WHITE WORM IS RELEASED AFTER 0.01% OF THE VULNERABLE SYSTEMS ARE INFECTED. THERE ARE 300,000 VULNERABLE SERVERS.**

Even with a supersensitive white worm, released after 0.01% of the systems are infected, most of the vulnerable population is still compromised by the malicious worm. This is because even with the highly (even optimistically) sensitive value of 0.01%, the black worm has infected 30 systems, giving it a nearly 5-generation head start over the white worm. If we even consider a perfectly timed white worm, equally matched to the malicious worm and released at the same time, the white worm is still expected to save only 50% of the systems.

Another important aspect of relying on a white-worm-based defense is the accuracy of the underlying detection mechanism. It is understood in the worm community that accuracy is the most important issue in developing a successful worm-detection capability [19]. The cost of a false alarm varies depending on the response. The best false alarm rates reported are on the order of just over once per day [4,13]. A false alarm that causes a significantly disruptive event, such as a premade white worm, could be catastrophic.

## Disruption from a Worm

One problem with worms is they can be disruptive to the network. For example, an anti-Slammer, released at the same time as Slammer, would have proved equally disruptive (at least until the white worm stopped spreading). In fact, Welchia, owing to a faulty ICMP scanner, was far more disruptive to the network than Blaster, the worm it was displacing. Blaster's traffic was simply normal TCP, and its scanning rate was low. Welchia had a high-rate ICMP scanner that flooded the local network with unresponsive traffic that caused congestion, and it was Welchia (the alleged white worm), not Blaster, that disrupted the Navy/Marine Corps intranet [9].

Of course, even if a white worm is released before a malicious worm, the white worm may prove to be as bad as or worse than the disease. Assume that the patch deployed by the white worm requires a system reset, but the original worm does not affect system stability. In this case, the white worm's act of resetting a critical but infected system may be more damaging than an unchecked infection by the malicious worm.

Additionally, a single-packet UDP worm such as Slammer or Witty will naturally be disruptive to the network. It would require significant engineering to develop a congestion-sensitive UDP-based worm, and such a worm would be considerably slower than a malicious worm, which doesn't care about fair congestion control.

## The Difficulty of Control

Even if a white worm is to be used for another task, such as proactively patching before a malicious worm can spread, there is still the substantial difficulty of controlling the system.

If the white-worm author doesn't a priori know which systems should be compromised and have provable coverage only over those systems, an error in this logic could prove catastrophic. The original worm paper by Shock and Hupp [12] only touched on the real limitations: the havoc caused by bugs in the worm that either would cause self-propagation (their model was mobile without duplication) or could disrupt the entire network.

The one proposal that attempts to address this, Nematodes, postulates either an end-host *allowed* token or an authorization server (in either case patch management could be deployed instead with much lower risk), or a *white graph* of allowed traversal topology.

But what happens if there is a failure (i.e., bug or unforeseen circumstance) in the white graph? For example, the Nematodes proposal remarks that attacking 192.169/16 space should be OK if the worm is already in 192.169/16 space (thus the notion of a white graph instead of a list). Yet what happens if happenstance causes a nematode to end up on a critical system in someone else's private address space (e.g., spreading via a laptop that changes locations and IP addresses)?

This is far worse than the problem of a bad input into a vulnerability scanner. A nematode, by infecting a host, may disrupt the host considerably more than a vulnerability scanner would. Worse, because of the self-propagating nature, a badly targeted nematode could spread to many more locations. With a bad entry in a vulnerability scanner, only those hosts affected by the entry will be discovered. But a bad entry in a nematode's target database could propagate, affecting far more systems.

## The Legal Minefield

It is this difficulty of control that brings up the greatest problem. If the white worm's author has legitimate access and control of the systems, he or she can use a patch-management tool. But if the author does not have legitimate access and control of the systems the white worm ends up infecting, the worm's author or releaser just committed a crime (a felony in the United States).

Even a nationally authorized white worm could encounter these difficulties. If the worm itself infected a computer in a different country, the laws of the computer's location, not of the worm author's, apply. If the breach of the white worm or the payload is damaging enough, the victim nation may perceive it to be an act of war and retaliate commensurately [18]. The potential for catastrophic political damage, even for a governmentally authorized worm, is difficult to understate.

## The Alternative: Patch Management

For networks that are controlled, patch management and system attestation provide all the benefits of a white worm with a much lower risk profile. System attestation (such as Cisco's NAC [3]) prevents any system from connecting that is not running a set of administratively mandated software, such as antivirus and patch-management tools. A patch-management system allows the administrator to push code to the systems dynamically. Using these two capabilities together, it is reasonable to assert that hosts are attested and up-to-date with the most current patches, assuming they haven't already been compromised and rootkitted.

Unlike a white worm, attestation and patch management has perfect and controlled coverage: All systems on which the patch-management software is installed, and *only* those systems, are updated. There are no issues with self-propagating code escaping into the wild. Patch-management systems are much easier to test than a white worm, as their behavior is more deterministic. Finally, patch-management systems can be intrinsically faster than a worm, as there is no target discovery, TCP sessions can be preestablished, and, with multicast, the same data can be sent to all systems simultaneously.

Patch management, however, is not without cost. It may require additional software, testing infrastructure, and sitewide policies to deploy. However, a well-run enterprise already needs a patch-management system to handle the large number of upgrades and patches for a variety of software packages. It seems far more reasonable to invest in extending that capability than it is to invest in a technology whose potential negative impacts are practically impossible to bound.

## Conclusions

The theme of using white worms has recurred roughly annually over the past few years as one proposal to deal with network worms. Network worms are a serious threat because they can spread quickly and deploy an arbitrary payload. It is natural to want to harness that power to do something useful. One naive proposal is to use them to counter other worms. However, the potential negative consequences of a white worm solution are exceptionally grave, because they cannot be practically bounded. Further, a patch-management system out-competes white worms in terms of performance (speed and coverage) and has a far more acceptable risk profile. The capabilities are also technically more mature and cost-effective than those of white worms. We, therefore, call on the community to discourage the consideration of white worms and focus intellectual effort on the many other hard problems in computer security.

### REFERENCES

[1] D. Aitel. "Nematodes—Beneficial Worms": http://www.immunityinc.com/downloads/nematodes.pdf.

[2] F. Castaneda, E. C. Sezer, and J. Xu. "Worm vs. Worm: Preliminary Study of an Active Counter-Attack Mechanism," *Workshop on Rapid Malcode (WORM)*, 2004.

[3] Cisco network admission control: http://www.cisco.com/en/US/netsol/ns466/networking_solutions_package.html.

[4] D. Ellis, J. Aiken, A. McLeod, D. Keppler, and P. Ammann. "Graph-Based Worm Detection on Operational Enterprise Networks," Technical Report MTR 06W0000035, The MITRE Corporation, April 2006.

[5] R. Foulkes and J. Morris, "Fighting Worms in a Large Corporate Environment: A Design for a Network Anti-Worm Solution," *Virus Bulletin International Conference,* New Orleans, 2002, pp. 56–66.

[6] A. Gupty and D. DuVarney, "Using Predators to Combat Worms and Viruses: A Simulation-Based Study": http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1377222.

[7] Herbert HexXer, Codegreen beta release: http://seclists.org/vuln-dev/2001/Sep/0000.html.

[8] I. Marson: http://www.zdnet.co.uk/news/internet/security/0,39020375,39182954-1,00.htm.

[9] E. Messmer, "Navy Marine Corps Intranet Hit by Welchia Worm": http://www.nwfusion.com/news/2003/0819navy.html.

[10] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the Slammer Worm," *IEEE Security & Privacy,* pp. 33–39, July/August 2003.

[11] D. Moore and C. Shannon, "The Spread of the Witty Worm": http://www.caida.org/analysis/security/witty/.

[12] J. Shoch and J. Hupp, "The 'Worm' Programs—Early Experience with a Distributed Computation," *Communications of the ACM,* March 1982.

[13] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated Worm Fingerprinting," *6th Symposium on Operating System Design and Implementation (OSDI '04)* (Berkeley, CA: USENIX Association, 2004).

[14] S. Staniford, V. Paxson, and N. Weaver "How to Own the Internet in Your Spare Time," *Proceedings of the 11th USENIX Security Symposium* (Berkeley, CA: USENIX Association, 2002).

[15] Symantec. W32.blaster.worm: http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html.

[16] Symantec. W32.welchia.worm: http://www.symantec.com/avcenter/venc/data/w32.welchia.worm.html.

[17] S. Tanachaiwiwat and A. Helmey, "Vaccine: War of the Worms in Wired and Wireless Networks": http://www.ieee-infocom.org/Posters/1568980643_VACCINE%20War%20of%20the%20Worms%20in%20Wired%20and%20Wireless%20Networks/Tanachaiwiwat_INFOCOM_abstract.pdf.

[18] W. G. Sharp, *Cyberspace and the Use of Force* (Ageis Research Corp., 1999).

[19] N. Weaver, D. Ellis, S. Staniford, and V. Paxson, "Worms vs. Perimeters: The Case for Hard LANs," in submission.

MICHAEL B. SCHER

# on doing "being reasonable"

Michael Scher is general counsel and compliance architect for the Chicago-based IT security firm Nexum. An attorney, anthropologist, and security technologist, he's been working where the policy tires meet the implementation pavement since 1993.

*mscher@nexuminc.com*

**IT'S AN ADMIN'S WORST NIGHTMARE:** Mission-critical servers are compromised, and you didn't know about it. Systems you can't patch except in tightly controlled downtime windows have nefarious changes being made to them by persons unknown; systems where uptime equals money and downtime equals loss; systems you probably could have patched sooner, but the politics were just too treacherous. Worst of all, you didn't even discover the intrusion—you found out from the legal department: Your company is being sued because those mission-critical, fiscally sensitive hosts are being used to attack a third party. That third party cried havoc and let slip the lawyers of tort.

Well, that's not the way it went down at IBM this summer [1], but a glimmer of that kind of scenario briefly made the news with IBM in the headlines. In short, a D.C.-area law firm sued Big Blue because, allegedly, a number of servers at IBM's Durham, N.C., facility were being used to attack the law firm, which had, allegedly, suffered harm as a result. IBM says the firm hasn't even demonstrated that either organization's systems had been compromised, let alone suffered harm, and has asked the court to throw the case out. We'll have to wait to see where this one goes, but even if it's a tempest in a teapot, it illustrates an emerging consciousness among the legal community that harm caused by way of an ill-protected computer can be worthy of a lawsuit.

Are you responsible for critical or Internet-facing systems at a large organization? Have you talked lately with corporate legal about the company's policies for systems security? Oh, sure, your group has probably discussed SarbOx, HIPAA, GLBA, the various state consumer information privacy acts (starting with California's SB1386), and maybe even 21 CFR 11 with them, inspired by auditors and under the shadow of possible penalties. Maybe your organization has a policy to cover and look for contributory copyright infringement, for uses that constitute harassment, or for communications that create a hostile workplace. But how about good old negligence? Now's a good time to have that talk.

Some of this paper will strike systems personnel as heavy on the legal material; it will strike attor-

neys as light on the legal side and full of too many technical details. My goal is to raise awareness about negligence law in systems groups and to help them start a dialog with their own legal departments or outside counsel. The content and positions contained in this article should not be taken as legal advice—the discussion is simply far too general to safely use that way. The purpose is to make you more conversant in the issues and able to discuss them with personnel responsible for corporate, fiscal, and legal risk.

## The Hypothetical: Arnold's Widgets v. Burt's Technical Company

Let's take a look at the kind of lawsuit in question. We'll walk through a hypothetical set of facts, and see where and how a negligence suit could fly. Since the facts are ambiguous from the recent story in the news, we'll start fresh. Let's say a company, Arnold's Widgets, has had some of its Internet-facing hosts compromised and others hit with traffic-flood denial of service (DoS) attacks by persons unknown, who apparently used compromised servers at Burt's Technical Company (BTC) to do their dirty work.

Arnold's has suffered several days of DoS attacks that rendered its widget-ordering site unavailable. The company estimates a nonrecoupable loss of business (which went to the competition) amounting to $300,000. Arnold's has also spent some $50,000 on forensics and consultants to locate and clear compromised hosts in its DMZ. The forensics folks identified several IPs in the BTC netblock as the source of both the DoS traffic and the direct system compromises. Arnold's sues BTC for damages of $350,000, which Arnold's claims are the result of BTC's negligence.

## Negligence

Negligence is at once a simple concept and a complex morass of subtle rules brought about through case law and statutes. Actions for negligence are part of "tort law"—non-criminal law handling harms caused by breaches of duty between two parties. At its core is the following principle: If you fail to live up to a duty, you're negligent. It gets a little complicated from there. There are a number of ways a person can be under a duty, but we'll just look at two:

- Duty to conform to regulatory or statutory requirements
- Duty to act reasonably

### DUTY TO CONFORM TO REGULATORY OR STATUTORY REQUIREMENTS

This principle seems self-explanatory. If there's a law or a regulation, and you don't live up to it, you could be seen as negligent. Indeed, in some states, it's a special kind of negligence called *per se* negligence. In essence, that means that it's presumed to be negligent to fail to live up to it—the defendant has to show that *not* doing the duty was reasonable. Normally, the plaintiff (the person suing) would have to show it was an unreasonable act, but *per se* negligence turns that on its head. Sometimes, the regulation or statute specifies what the injured party can collect ("a remedy"); in some circumstances, that means the statutory or regulatory remedy is exclusive, and one cannot sue otherwise over the harm caused by negligence; one can just request whatever remedy is specified. Sometimes, the remedy is not meant to be exclusive—some provisions don't have a specified remedy at all, leaving "enforcement" mostly up to the public, who may sue for harms resulting from failures to obey the law. Politicos don't have to show much of a budget to enforce laws or regulations that will be

enforced by the public through lawsuits (though certainly it does cost the system money).

## DUTY TO ACT REASONABLY

This duty can be more complicated and lies at the core of the more "interesting" lawsuits alleging negligence. As a member of society, or of some specific profession or subgroup, you have a duty to act like a reasonable person, taking reasonable care in your behavior to avoid reasonably foreseeable harm to a reasonably foreseeable swath of people. Doctors have to act like reasonable doctors when doing doctor stuff; systems administrators have to act like reasonable systems administrators when doing sysadmin stuff; and the person driving a car has to act like a reasonable car driver while driving. Sounds reasonable, right [2]?

## DETERMINING WHAT'S REASONABLE

We already discussed how, usually, it's considered reasonable to follow the directives of law or government regulation, and how in some jurisdictions there's a presumption that failing to do so is presumed to be negligent. So where, for example, HIPAA lays down some rules, it would be wise to follow them, not just for HIPAA reasons but because not doing so may be construed as unreasonable behavior. But what about when there's no specific law or regulation? I mean, most of us think about lawsuits as the result of mistakes of some kind or clumsy error, so there's got to be some kind of reasonable standard, right? Right. Kind of.

Reasonableness is determined by the finder of fact—a jury quite often, and sometimes the judge—based on community standards of behavior. Usually, that means, in the absence of a law or regulation, you can look to common practice for a decent idea of reasonable behavior. But not always. "This is the way we've always done it" is usually disingenuous, and in any event, the excuse doesn't always hold water.

In a famous case from 1932, two barges sank in a storm while being guided by tugboats, and the tugboat operators were sued for losing the barges. Neither tug had a radio, so they failed to receive warning of the oncoming storm and taken shelter, as many other tugboats did that day. The industry was just starting to adopt radio receivers in boats and it could hardly be called the norm to have them at the time. The tug owners were found to be negligent for not having radios, and they had to pay for the barges, despite doing what many if not most tug companies were doing. In his decision affirming the judgment, Judge Learned Hand set down an important principle [3]:

> Indeed in most cases reasonable prudence is in fact common prudence; but strictly it is never its measure; *a whole calling may have unduly lagged in the adoption of new and available devices . . . there are precautions so imperative that even their universal disregard will not excuse their omission.* [emphasis added]

In IT, and all other fast-changing industries, that means in essence that one may want to look a little bit ahead—in short, at *best practice*—to help determine what's reasonable. If achieving best practice levels is *reasonably* cost-effective, it might not be reasonable to slouch along with the competition. Those lagging behind may be deemed negligent.

Another famous Learned Hand decision, again involving barges, came in 1947. Tugs and barges seem to have been on the technology and risk fron-

tier of the day, an industry trying to compete and cut costs by saying "this is the way we've always done it!" His decision will sound familiar to anyone familiar with risk analysis, in IT or in tugboatdom. This time, a barge broke loose in a storm, smacking into other vessels at dock, causing a lot of damage. No attendant was kept stationed at the barge because it would have been too expensive. Judge Hand wrote [4] that the level of reasonable care can be estimated from:

> (1) the probability that [the risky event will occur]; (2) the gravity of the resulting injury, if [it] does; (3) the burden of adequate precautions. Possibly it serves to bring this notion into relief to state it in algebraic terms: If the probability be called P; the injury, L; and the burden, B; liability depends upon whether B is less than L multiplied by P; i.e., whether B < PL.

Sounds reasonable, right? If the risk is caused by your operations, and the burden of protection is not so bad, and the potential harm is significant and not unlikely, then there's a duty as a reasonable person to avoid subjecting others to that risk.

Most states also have an inverse rule: The victim can be penalized if his or her *own* negligence added to or helped cause the harm. In many states, the ability to collect is completely cut off if the harmed party was more than 51% responsible, as determined by the fact finder. In a handful of states, the harmed party can't collect if they were even a little responsible, and in all the rest, some balancing act takes place to allocate the costs of fixing the damage among the parties based on comparative fault.

## DO WHAT YOU SAY AND SAY WHAT YOU DO

Does your organization have any systems policies, designed to protect some group of people from some group of harms, which your organization isn't quite meeting? While probably not *per se* negligence, it's definitely good evidence that your organization knew what the right thing was, wrote it down, and then failed to do it. That is, it's evidence that your organization was negligent. That's a deep hole to explain your way out of when the harm the policy is supposed to have prevented comes to fruition.

Corporate legal should help the technical organizations craft policies that can actually be implemented on the ground in a realistic time frame. Don't set or accept pie-in-the-sky policies that can't be done. It's your job to talk with legal, and with corporate finance, to find that magic balance point of risk and the cost to avoid. If there's a need to reach some policy directive, but it will take time, make your policy actually set a time frame. If it looks as though you will miss the due date, ensure policy is changed to a later due date long before it's reached. It's better to try again honestly than to live with a policy level you're simply not meeting.

## COLD HEARTS AND RISK ANALYSIS

There are of course circumstances where the harm is so significant that calculating it away as too costly to avoid will be frowned upon in the courts. Matters of life and death, or broad health disaster, or the performance of generally life-threatening acts, or allowance of obviously life-threatening problems under your control will probably not be considered reasonable and may be the subject of punitive damages [5]. Probably you won't see much of that in systems administration, but it would certainly be wise to ensure the controls to your medical therapeutic systems aren't available

from the Internet. In general, one should consider any precaution priced a lot lower than the potential harm.

At some level of inherent risk, the courts are likely to say that, if it's so expensive to fix, and so risky to do, it probably should not be done, unless one is willing up-front to take on the costs of harm. For example, there are practices that are so inherently dangerous that a standard called "strict liability" applies. That is, the practice is so dangerous that the risk of harm cannot be eliminated by any expense; therefore the organization so doing is simply liable for the consequences—costs to avoid and standards of care are irrelevant. Blasting and some aspects of defective product liability are, for example, held to strict liability standards [6].

### HARM AND "ACTIONABILITY"

Here's a funny thing: if someone is completely negligent, but no one is harmed, then no one gets to sue for negligence. There has to be a harm. Tort law is there to make the harmed party whole—no harm, no tort. The harm also needs to be one that is reasonably foreseeable, the result of the failure to meet the duty, and of the general kind that the duty was there to prevent. It can be a surprising harm in its scale, so long as it's a direct result of the failure to meet the duty and a foreseeable kind of harm. That set of relations from an act makes that act the "actual and proximate cause"—that's a key (and sometimes quite complicated) concept in practice.

Another principle is that sometimes an intervening action by a third party can cut off your liability for failing to meet a duty. Their action must constitute a "superseding" cause: it has to occur after your negligence, and become one of the proximate causes of the harm. This principle holds mostly when the intervening act was itself not
a reasonably foreseeable circumstance, so that you could not reasonably be expected to take it into account in forming your behavior. So if you're a landlord in a bad neighborhood and the tenants complain that there is a dark spot at the entry that muggers can use to attack them, and you do nothing about it, the mugging is probably not going to be seen as an intervening act: It's the very act you were warned about and part of your duty as a landlord to mitigate. Mind, if you took every reasonable step and someone still committed such a criminal act, you would probably not be viewed as negligent and the act would be seen as intervening.

Whew!

## Summing It Up for Our Theoretical Arnold's v. BTC Case

Arnold's has alleged the following:

- BTC has a duty to keep its servers reasonably secure against unauthorized access; alternatively, if an employee failed to perform his or her duty securing the hosts, BTC nevertheless has a duty to supervise its employees.
- BTC failed to act reasonably as is evidenced by someone using its servers to attack the Arnold's network.
- BTC is responsible even if the attacker was an unauthorized attacker who illegally gained access to the BTC servers.
- Arnold's has suffered actual fiscal damage as a proximate result of these attacks.

BTC replies:

- BTC takes reasonable steps to secure its hosts, based on industry-normal practices and recommendations from its auditors. It looks to its industry for guidance.
- The break-in occurred despite reasonable practices.
- The intervention of a criminal act cuts off the proximate causation of BTC's alleged failure to meet a duty and places the fault strictly on the criminal.
- The compromised servers at Arnold's were compromised only because they themselves were not well secured, unreasonably so, to no less a level than BTC's alleged negligence, and so Arnold's is contributorally negligent.

Let's go through it by the numbers.

1. Did BTC have a duty to keep its hosts secured to some level? Does such a duty exist? Well, we all know a compromised host can be used to attack others and cause harm. Compromised hosts also help shield the identity of an attacker using them to go after third parties. That in turn hurts the third party's ability to sue the real attacker. Is it reasonable, knowing all that, to set up a situation where attackers are *likely* to be able to use your servers to attack others? Probably not. It seems likely that such a duty could be found to exist, a duty to keep the hosts secured to a reasonable level. If BTC had a policy saying it had to keep its hosts patched up to snuff, and failed to meet its own policy, that's some more evidence that BTC didn't live up to a reasonable standard of behavior—one it had set for itself, no less. BTC could claim its own policy was unreasonable, but let's face it—that isn't going to sound good in court.
2. Were BTC's practices reasonable? Did BTC fulfill its duty to act reasonably? Most companies do make the effort to firewall, watch, and patch their hosts, especially the Internet-facing ones. If nevertheless many get broken into on a recurring basis, common practice may not be enough to be reasonable. "Reasonable" behavior may be to move toward protection levels that more or less cut off what we all understand to be *one of the most likely* (and therefore foreseeable) outcomes of a host compromise. So if BTC's actual practices were below what (many expensive) experts say is "common practice" for the industry, it could well be found negligent. If it acted at "common practice" levels but below "best practice" levels, it could still be found negligent, a rather damning message to the industry in question. The question would be in essence: What does the company need to have done to constitute "reasonable" steps to cut off such a likely harm?
3. Did the criminal actions of the attacker cut off BTC's responsibility? Consider that, but for the actual criminal actions of the attacker, there would have been no harm to anyone, let alone to Arnold's. Yet, an exposed system, we are all aware, would probably be probed by automated attack processes several times a week, if not more frequently. Studies show that they are subsequently compromised on an ever-more-frequent basis [7]. Was it inevitable or at least reasonably foreseeable that a host with a vulnerability would be compromised (criminal act) and used for nefarious purposes (another criminal act)? I think we can all agree that, yes, it is likely. Is the action, therefore, more like a mugger in a dark spot that is plainly useful to muggers and about which the landlord has been warned than like a burglar using top-of-the-line tools to break into an ordinary apartment? I think we can agree that a key point of border security and patching is to prevent

exactly this kind of criminal act, so it would be likely that the act would not cut off liability. It's the very series of events the duty would be there to prevent.

4. Was the negligence the proximate cause of the harm? If (given all the above) the attacker's act is held to be just part of the stream of reasonably foreseeable events, then BTC's negligence is a proximate cause of the harm. Arnold's wins! Well, maybe not. BTC alleges that Arnold's was at least partly responsible for the harm by *not securing its own machines*—the very same duty BTC failed to perform. Here it looks like BTC finally has some traction: those servers that were compromised were certainly not up to snuff, and an expert could argue that they would have been broken into eventually. Depending on the state law, Arnold's could be in a position to collect nothing for its damages in finding and fixing the compromised hosts. Applying comparative negligence to a DoS attack, however, is harder. It's not currently even best practice to be truly DoS proof, lots of products claiming to help at low cost notwithstanding. It costs a lot to defend against a traffic flood on your own, and having the resources to do so is a business decision mostly brought about by frequently being in the sights of DoS-based attackers. Then again, good contracts with upstream providers, good upstream providers, and emerging services that significantly reduce the effects of a traffic flood DoS will shift reasonable DoS protection from the rare and extreme to the best practice realm probably within a few years. Only companies requiring instant defense (rather than within an hour from a service provider) might require a higher standard. Which way the jury might go on this last score will be the result of the expert testimony. Nowadays, not noticing that your servers are flooding someone else's network could tend to be seen as negligent. Conversely, I'd guess most businesses don't need extreme DoS defenses to be "reasonable" but someday, not too far away, they well might.

## Hypothetical Case Summary

BTC is probably going to have to pay out something to Arnold's for the harm caused by the DoS flood. Yet, the entire experience was probably stunningly expensive for both companies. Obviously, they might well settle long before reaching the trial stage. Nevertheless, one doesn't want to be in BTC's position. And one never wants to be on the receiving end of a landmark case.

You're probably not negligent if you act with reasonable, due care, even if harm somehow happens. You're probably not liable (though you would be negligent) if your failure to act reasonably results in no harm, or harm of an utterly different kind than that which the duty to act reasonably is there to prevent. It's not an excuse much of the time that the harm depended on a criminal third party if the third-party action is reasonably foreseeable and the harm is the kind you'd expect.

## Compliance Is a Two-Way Street

Systems and security/risk teams need to talk to corporate legal and business risk personnel. Instead of just implementing controls that react to compliance regimes, systems and network security needs to work hand in hand with legal in the crafting of security policies. The issues aren't just that the company will itself be directly harmed (risks that a good risk man-

agement group can estimate) but that the company's security posture decisions could result in harm to foreseeable third parties.

And hey—let's be reasonable out there.

## REFERENCES


[1] "D.C. Law Firm Claims IBM Worker Hacked Its Computers": http://www.informationweek.com/security/showArticle.jhtml?articleID=190400233.

[2] We'll just pretend we didn't all just think "and what about the reasonable attorney," for now. OK? Good.

[3] The T.J. Hooper, 60 F.2d 737, 740 (2d Cir. 1932).

[4] U.S. v. Carroll Towing Co., 159 F.2d 169, 173 (2d Cir. 1947).

[5] Although the famous Ford Pinto cases from the 1970s are often discussed as exemplifying this principle, they are actually not very good examples. See "The Myth of the Ford Pinto Case," by Gary T. Schwartz, 43 Rutgers L. Rev. 1013 (1991), available at http://www.pointoflaw.com/articles/The_Myth_of_the_Ford_Pinto_Case.pdf.

[6] *Inc.* magazine had a 1999 article with a number of strict liability examples. It's online at http://www.inc.com/articles/1999/11/15396.html.

[7] See, for example, the SANS ISC study: http://isc.sans.org/survivalhistory.php.


## OTHER RESOURCES

Carter Schoenberg of ISS has written an excellent primer on the relationship of regulatory processes and patching to negligence: http://www.infosecwriters.com/text_resources/pdf/InformationSecurityCClass.pdf.

A discussion of the differences among contributory, comparative, and modified negligence, and a listing of which states follow which principle, can be found at http://www.mwl-law.com/PracticeAreas/Contributory-Neglegence.asp.

MIKE HOWARD

# how often should you change your password?

Mike Howard came into programming from systems engineering and has been stuck there. He currently makes his living doing custom software and system administration for a few small companies.

*mike@clove.com*

**FOLKLORE TELLS US THAT WE NEED** to change our passwords fairly frequently. In fact, it is required by the security policy of many companies. I recently revisited the problem and have concluded that *changing passwords doesn't really matter*.

The issue came up because a client for which I have worked for about twenty years recently sold two of its divisions to a major company. The new owners—as you would expect—have all kinds of security requirements and security specialists and policies and what not. In contrast, our security policy was almost nonexistent, not enforced, not policed, and implemented only on privileged accounts. In spite of this, we have never been successfully penetrated from the outside.

Admittedly, the company is not a significant target, but looking at the *secure* logs on the Linux systems on our public networks revealed that they had been subject to fairly continuous, automated, low-level probing. I once made the mistake of misconfiguring an Apache server so it could become an open mail relay and that was almost instantly discovered and used—so automated probing is effective.

During the time we had public networks—well over eight years—we had only changed the root password once and had had that same administrative password on all the servers. We have had dial-in connections for well over twenty years without a break-in. (Although attack through a dial-in port seems unlikely, I personally learned early on that even an unlisted phone number is not safe: I had an early Xenix system cracked when I sloppily configured it *without* a root password.)

So I got curious.

This note is a summary of what I came up with. A more detailed note, "How often should you change your password—or should you bother?" is available at www.clove.com/clove_tech/tech_notes.

Throughout this note, I use the letter $k$ to denote the cumulative number of password crack attempts, $N$ for the total number of probable passwords, and $P_k$ for the probability of being cracked after $k$ crack attempts.

## Effect of Changing Passwords: $P_k$

If you think about it for a minute, cracking passwords is a classical "drawing with (or without) replacement" probability problem. If I have a sack containing $N - 1$ black balls and a single white one, then guessing a password is equivalent to pulling a ball out of the sack. If I leave the ball out, then that is the same as not changing my password. If I put it back each time, then that is changing my password after each attempt.

In the first case, $P_k = k/N$.

In the second case, $P_k = 1 - (1 - 1/N)^k$.

For anything else, the probability must be someplace in between.

I made the reasonable assumption that we want to keep $k << N$—that is, we want our number of crack attempts to be much less than the number of probable passwords. If this is true, then we get a good approximation for the "continuously changing password" case of $P_k \approx k/(2N)$.

So, in general $k/(2N) \leq P_k \leq k/N$, where you need to take the left $\leq$ with a small dose of salt.

So, no matter how often you change your password, it doesn't have much effect on the $P_k$—the probability that your password will be cracked.

All that matters is keeping $k$ small relative to $N$. In other words, we want $k$ (number of crack attempts) small and $N$ (number of probable passwords the cracker has to probe) large.

## Probable Number of Password: *N*

The number of probable passwords a cracker has to test is under the control of the users. That is, we usually pick our own passwords. As a result, passwords are generally constructed of words, pseudo-words, and numbers. A few brave souls add punctuation and other things, but my sense of things is that this is rare. For the most part, we use words.

This is important, because our use of words severely limits the size of $N$. For example, if we construct five-character passwords from the lowercase alphabet alone, we will have $1.2 \times 10^7$ possible symbols. Words, however, are more restrictive.

To get a handle on this, I downloaded the *King James Bible*, *The Federalist Papers*, and *A Short Biographical Dictionary of English Literature* from the Gutenberg Project and wrote a little code to do a naive analysis. It turned out that the three texts only contained $10^6$ words, of which $2.7 \times 10^4$ were distinct. In fact, there were only 2,283 five-character words and only 3,271 eight-character words.

It is clear that using words is dangerous, because $N$ is far too small.

In our case, we generate passwords using a system based on folklore:

- We create a five- to eight-character alphabetic string by selecting a single word or concatenating two short ones together.
- We then randomly capitalize a few letters.
- Finally, we insert a couple of digits.

When I analyzed this scheme using the words in my aforementioned documents, it expanded the number of eight-character passwords from $3 \times 10^3$ to $8 \times 10^{12}$. If you are willing to use ten-character passwords, this goes up to $7 \times 10^{14}$. Again, see "How often should you change your password—or

should you bother?" for more details at htttp://www.clove.com/clove_tech/tech_notes.

The Teracrack project [1] used a dictionary from "crack" containing about $5x10^7$ passwords to create a precomputed password hash of about $2x10^{11}$ entries. Briefly, the Teracrack project tested the feasibility of speeding up cracking by precomputing password hashes used by the crypt algorithm. The project was able to demonstrate that crypt is unsafe because the hash database fit within 1.5 terabytes of disk—an amount easily affordable nowadays. If the size of the password hash scales roughly linearly, then this algorithm should expand the hash requirements by five orders of magnitude—say, to something on the order of $10^{16}$ for eight-character passwords—and so require something on the order of $1.5x10^5$ terabytes. This is a fairly simple change which should move the storage requirements of this technique out of reach for a little longer. This is *not* an endorsement of crypt, which has been known to be deficient for at least twenty years.

Variations of this scheme yield different effects. Here are some rules of thumb:

- Adding one character of password length increases $N$ by a little less than a power of 10.
- Inserting an additional digit increases $N$ by about a power of 10.
- Replacing letters with digits helps a little, but much less than insertions.

Our experience is that passwords of this form are fairly easy to type from memory—after practicing a few times—even though it is often difficult to remember how to write them down with a pencil.

As a side project, I looked at the effect of case-insensitive password schemes—which are common with certain large computer manufacturers. In general, they reduce the number of probable passwords using this scheme by about two orders of magnitude for short passwords and three for longer passwords (of length 11 or 12).

Case sensitivity in password schemes is very important.

## Controlling *k*

The value of $k$ is not static. It grows as crackers attempt to crack and system administrators and system software vendors do nothing about it. It has nothing to do with users and their changing of passwords.

Crackers appear to be building nets of robots that mechanically attack systems. There is not much we can do about that other than detect crack attempts and shut them down.

I made some off-the-cuff computations and came up with an estimate of $3x10^8$ crack attempts per year per host by assuming that the system is exposed to one crack attempt per second. This gives a $P_k$ of about 0.0001 per year for an eight-character password.

Referring back to why I started thinking about this in the beginning, we see that, in our case, we had seven hosts on the public net and experienced nowhere near that crack attempt rate, so this explains why we were not cracked. Our $P_k$ was well below 0.006 [0.0001 x (7 hosts) x (8 years)], which is pretty good odds of not being broken into.

To digress for a moment, we and crackers have different points of view. From our point of view, we don't want to be cracked, so these are fairly good odds. However, from most crackers' point of view, they are also good

odds because he or she probably just wants to crack some system—not necessarily a particular one. So if he or she attacks 100,000 systems in an automated way, then one should burst open often enough to be interesting. This is kind of a sick win-win situation for both sides.

I don't want to have one of those systems, so I did the obvious thing—devise a way to detect crack attempts and cut them off. If I cut off hosts that attempt to crack my system more than, say, five tries in an hour, then the cracker needs to control a vast number of systems to be effective against me.

To do this, I wrote a short hack which monitors /var/log/secure to detect crack attempts on *ssh* and firewall-off suspicious hosts. The code requires Python 2.4, is GPL'ed, and could easily be adapted to monitoring for other types of crack attempts—say, against Apache servers. Again, see www.clove.com/clove_tech/download/.

## Conclusion

Changing passwords frequently is a doomed strategy.

The important thing to do is keep $P_k$ small by bounding the rate of growth of $k$ and constructing passwords so that $N$ is large, both of which are easy to do.

## REFERENCES

[1] T. Perrine, "The End of crypt() Passwords . . . Please?" *;login:* (December 2003): 6–12.

MARK BURGESS

# configuration management: models and myths

## PART 3: A SHOCKING LACK OF AD-HOCRACY

Mark Burgess is professor of network and system administration at Oslo University College, Norway. He is the author of cfengine and many books and research papers on system administration.

*Mark.Burgess@iu.hio.no*

**IN HIS 1970 BESTSELLING BOOK** *Future Shock* [1], writer Alvin Toffler predicted the demise of bureaucracy. Toffler was a writer emerging from the 1960s, on the tail end of the hippie revolution. They were going to make the world right, optimism was in the air, and everyone saw the pace of technological change as a force for good. Today, we are less enamoured by progress and have fallen back into a stagnant economic tumble-drier of selling and consuming that seems to have no vision or direction. Perhaps that is why Toffler's vision of the demise of bureaucracy never really came about.

Toffler predicted that centralized power structures, with their rigid procedures for decision-making and management, designed for a slower age, an age of little change, would collapse under their own sluggishness—buckling under the force of a cultural and technological deluge. Bureaucracies would be replaced by lean, mean decision machines, guided by simple principles, and so agile that they would win over traditional leviathans, much like mammals sticking out their tongues at the sauropods. Moreover, people like me, working in government organizations, would be freed from the slavery of application-report-archive to live productive lives full of choice and measured reflection. He called this state of affairs ad-hocracy. Toffler wrote:

> Faced by relatively routine problems, [Man] was encouraged to seek routine answers. Unorthodoxy, creativity, venturesomeness were discouraged . . . rather than occupying a permanent, cleanly-defined slot and performing mindless routine tasks in response to orders from above, [Man] must [now] assume decision-making responsibility—and must do so within a kaleidoscopically changing organizational structure built upon highly transient human relationships.

That is what Toffler said about the human workplace in 1970. This well-meaning sermon has admittedly not taken the human world by any great storm, as we attest from experience (though, if we are being fair, it has indeed made inroads). What I find ironic is that we are now reliving an almost identical discussion in a different sphere. Today, we are struggling to accept the same wis-

dom in the area of computer management. It will take the next two parts of this series to do this subject justice.

## Strategies in the War Against Tera

What is a good strategy or algorithm for computer management? Few would argue against the idea that the sheer size of systems today practically necessitates automated tools. (Recall Ken's law: Always let your tool do the work.) Certainly I believed this in 1993 when I started writing cfengine, and today IBM certainly believes it and flags it with its Autonomic Computing initiative. Toffler pointed out that automation does not necessitate production-line thinking, in which one mass-produced identical copies—a world in which one can have any color as long as it's black. On the contrary, he argued that "As technology becomes more sophisticated, the cost of introducing variations declines."

But in the management of the information technology itself, we are still hearing about "ways to mass-produce 1000 workstations, all identical, from a common source"—golden master servers that are to be worshipped by hundreds, perhaps thousands, of clones. Ad-hocracy is not the default doctrine in computer administration.

Ever since the late 1980s, the telecommunications companies have had their own vision of computer resource management, borrowing from tried and trusted inventory systems, for warehouse and personnel management, and trying to modify them to cope with the computing age. In time they borrowed ideas from software engineering (e.g., object-oriented database models).

Industry standards organizations such as the renamed Telemanagement Forum (TMF) and Internet Engineering Task Force (IETF) have continued to develop models for managing computing equipment that are essentially bureaucratic. What they perhaps failed to anticipate was the pace at which the technology would develop (something akin to the rate at which device drivers have to be written on PCs). Trying to keep up with the schema-centric definitions for all new products has led to a classic "Tortoise versus Achilles" race between the development of new technology and the struggle to document the growing zoological inventory. (Cisco's IOS is surely the winner of this race.)

For the telecoms, Operational Support Systems (OSS) and Business Support Systems (BSS) were the order of the day. The idea was simply to document every device and human procedure exhaustively in a huge database so that help-desk staff would be able to see an overview. Later came tools that could interact with the devices via a "management console" in order to write certain values to routers and switches, and even to workstations and PCs. Today, the legacy of these approaches is still with us; they still cling to life, even today in the largest corporations, but they are still wailing (or yawning) from their tar pits.

The complexity of those systems is legendary. No sane engineer, in his or her right GHz CPU, would seriously try to build such a monster. Yet, in the wake of these support systems, designed for the telephone era, the same knowledge engineers attempted to create the new generation of forms and processes that would manage the computing age. Among today's species:

- *SNMP/MIB:* A hierarchical table-based data structure (the Management Information Base) that is mapped into a linear set of machine-readable

identifiers. The values associated with these identifiers are simply read or pushed into place by the SNMP read/write protocol. The algorithmic complexity is very low. The data complexity is a simple regular approximation to a context-free language.

- *SID:* Shared Information and Data model. This is an information model that is used in both NGOSS and DEN-ng. It includes services and organizational containers in an object-oriented framework.
- *CIM:* Common Information Model. An information model that provides an exhaustive replacement for MIB.
- *NGOSS:* The TMF describes this as a "comprehensive, integrated framework for developing, procuring and deploying operational and business support systems and software." It includes the SID and eTOM standards. It is a complete organization map.
- *DEN(-ng):* Directory Enabled Networks. This is a model that is complementary to SID. It focuses on modeling network elements and services, using an interpretation of policy-based management. DEN-ng products and locations are subsets of the SID.

I challenge readers to look up the data models on the Web to see just how complex they truly are. The DEN-ng and SID initiatives are trying to move away from a MIB-like catalog of device attributes to an overview of an organization and its resources. In particular, the notion of services is an important addition.

Even equipped with these big guns for pattern description, and having the most eager blue-collar beavers to register all of this information, the efforts of these engineers ultimately seem to have fallen on deaf ears. No one really seems to want these systems—not even their key designers. Why? As the Soviet Union or European Union or even the State of the Union will testify, bureaucracy is just too expensive.

## What's on the Yellow Brick Road?

The data models mentioned here have sufficient linguistic complexity to describe the patterns we would expect to manage in an organization, just as we predicted in the last issue's episode, but something is wrong. Toffler's warning is ringing in our ears. We seem to be missing a vital part of the story. Configuration management is not merely about brick-laying and form-filling.

Configuration management (a pretty low-level animal in the administrative phylogeny) has become the topic de jour in the UNIX world, perhaps because it is a technological problem, which tech folks love. But it is not the beginning or end of any story that we really care about. We have no real interest in what the configuration of a system looks like. What we really care about is how to represent the goals of our organizations and applications using patterns in order to lead to a predictable pattern of behaviors. This leads us to a hypothesis, which, as far as I know, has not been convincingly proven:

> Hypothesis: There is a direct association between a "correctly configured computer" and a "correctly behaving computer," where "correct" means "policy or specification compliant."

The essence of this hypothesis is shown in Figure 1. It is not just a matter of configuring a computer but one of solving the problem of achieving the correct behavior. Configuration is a static thing; behavior is a dynamic consequence, but not a fully predictable one.

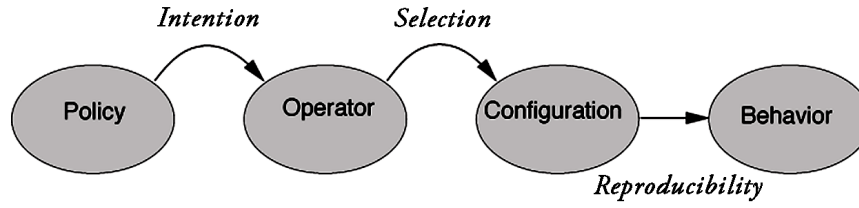



**FIGURE 1: THE STAGES FROM POLICY TO BEHAVIOR. A STORY QUICKLY FORGOTTEN?**

There are three parts to the story depicted in Figure 1::

- Planning the intended behavioral policies for all parts of a system.
- Mapping this to a configuration that can lead to the correct behavior.
- Implementing the change in configuration reliably.

How do we know that we can complete this manifesto? Is it doable? If so, can it be done reliably? Well, in 2003, I proved a limited version of this hypothesis [2], showing only that it is possible to define the meaning of "policy" in terms of configuration changes so as to lead to predictable behavior on average. This is not quite the same as the hypothesis posed here; what it says is that there a restricted language that maps directly to behavioral consequences, so if we restrict ourselves to that, we are okay. The part about "on average" is general, and it says that no configuration management scheme can guarantee that a host will always be correctly configured, unless the machine is never used.

## You Say Tomato and I Say . . . Semantics

According to the first two parts of this series, we have a reasonable account of how to manage patterns of data (with or without the monstrous data models that pepper the procedures with structural complexity). These procedures might be messy, but they are essentially just bureaucratic spaghetti, somewhat irrelevant to the deeper issues.

If we fix a bit string, such as a file mode, using a numerical value, there is little ambiguity in the procedure. It seems like a straightforward problem to write some configuration to a computing device: This is like stamping out molds from a production line (e.g., chmod 755 filename). It is straightforward, easy, and not complex—much like SNMP without MIB. Any complexity lies in the patterns themselves—in the coding of the instructions, and in understanding what the behavioral consequences of these changes are. Thus, this is not where the problem lies.

But now consider the expression of policy itself. If we wish to describe an operation in terms of a high-level procedure (e.g., "InstallPackage(ssh)") then this is no longer straightforward, because it is describing the configuration coding only at a medium level, not all the way down to the bits. This is like saying, "Make me prettier!" It is not a uniquely defined or reproducible goal. Someone might say that it is their policy to make you prettier, but you cannot guarantee their behavior from this assertion. (You might trust them more, if they told you about what end result they were going to guarantee—see the following.) If we take only a shell of patterns such as InstallPackage, there can be several (even many) nonequivalent ways of defining the internal procedures within the language of the low-level configuration. Consider the following two interpretations of an InstallPackage command, which are inspired by real examples:

```
InstallPackage(foo)
        Check dependencies
        Check if package README exists
         if (!exists)
                copy package
                unpack
                run local script

InstallPackage(foo)
        Check if existing binary is executable
        if (!exists_and_exectuable)
                Check dependencies
                Copy packages
                unpack all
                copy files to /usr/bin
```

The resulting patterns are described and implemented in terms of language syntax, as we have already noted, and computing is obsessed by syntax today—but if the complete syntax is missing from the explanation, the call for InstallPackage is meaningless. Several of the big data models mentioned here boast a specification written in the Unified Modeling Language (UML), which is based on an object-oriented syntax (i.e., hierarchical class structures). Thus it is fundamentally built as a bureaucracy of types. Moreover, XML has become the bureaucratic memo-paper of choice. XML is no more than an empty syntax "desperately seeking semantics."

This is pretty much what happens in configuration management tools. By attempting to be user-friendly and high-level, many configuration tools sacrifice operational clarity for human readability. Trying to define configuration in terms of such vague high-level precepts is like trying to tell a story like this:

> A man (motion-verb) into a (drinking-place-noun) and (communication-verb) a drink.

We can fill in many alternatives that lead to grammatically correct sentences, i.e., which obey a pattern language that is recognized by our system. But the patterns all mean quite different things, or perhaps nothing at all. There is no clear way to say that what we meant was "a man walks into a bar."

If we are to successfully govern systems, either externally or autonomically, we need to be able to complete the chain from top-level goals, to a clear and reproducible set of operations, to a definite configuration that leads to predictable behavior. This is not an impossible task, but it is far from guaranteed.

## How to Say What You Mean

At the 2001 cfengine workshop (later followed up by Paul Anderson and opened to a wider community, becoming the configuration management workshop), a discussion almost became an argument. My friend Steve Traugott, bless him, told me I was wrong. Thunderclaps sounded, screams were heard. Tempers were enraged. In the meantime, Alva Couch and I were quietly interested in Steve's point as others were doing battle over it. I thought, "Clearly, I was not wrong; I am surely never wrong," and yet Steve pressed his point, which has since been studied in detail by Alva Couch and which I have come to understand better as I have pondered the matter using different reasoning. Of course, neither of us was wrong, but, importantly, something was learned.

The matter concerned two design strategies that have been discussed for constructing configuration management schemes:

- We specify the final state and leave it up to the program to figure out the details of getting there.
- We specify the starting point and a specific program of steps to take.

For reasons we won't go into yet, these were labeled "convergence" and "congruence," respectively. To borrow Alva Couch's terminology, we can rather refer to these as precondition-based and postcondition-based specifications.

Ultimately, I believe that the first of these is preferable for a number of reasons, including parsimony, consistency, and aesthetics (stay tuned), but the real difficulties associated with configuration management are present in both cases. They cannot be avoided simply by choosing one.

In both cases there is the matter of how it is possible to change from the old state to the new state. Suppose a computing device is in a state that is not consistent with policy. We require a procedure, whether that means a static bureaucratic procedure or a lean-mean entrepreneur procedure, to fix it. In the first case (postcondition), we define this procedure generically, like a template, once and for all (i.e., we define what we want to get out of "make me prettier"). In the latter, we define the procedure in each case, making it potentially inconsistent. Steve said we can still achieve consistency by always starting from a known state and following a precise chain of preconditioned actions, meaning that if a computer gets messed up, you wipe it clean and start over. This is a reasonable approach to take if one thinks in production-line terms about configuration management, but this is not my vision.

## Mass Production Undone

Production-line factory thinking requires a chain of preconditions. When you create a chain of operations that depends on previous operations, each step is preconditioned on what came before. If one step fails to be implemented, all subsequent steps fail (e.g., "I'm sorry, sir; I can't make you prettier; your nose is in the way.").

This is fair enough—we just have to figure out how to get it right without getting stuck. That might be possible, but in fact it is harder than in the postconditional case, because the compositional complexity of the approach has to be dealt with in one go, whereas it only has to be dealt with for one operation with postconditions. But the real problem with preconditions is that the approach fails to easily support a wide variety of different adaptations. It takes us back to Toffler's fear of the totalitarian-commie nightmare of mass production of a single unvarying model.

Oddly, in system administration, many still worship the totalitarian gods of mass production. The god of small things, to paraphrase Arundati Roy, is still being trampled by the heavy boots of bureaucratic thinking.

Suppose we assume that the postcondition model is possible (cfengine uses this approach, so it works at least in some limited capacity). Then we can (at least try to) never base an operation on something that came before. Then the order no longer matters, and only the final state is significant. Now, although this approach is achievable, in principle, it is also beset with problems. Its chief selling points are:

- Consistent semantics.
- Specification of the final state is often simpler than specification of the steps needed to get there.

- You do not have to wipe out a machine if something goes wrong; the system can adapt in real time.

Its main problem is a residual ordering ambiguity caused by creation and deletion and competitive adaptation.

## Black Boxes and Closures

The inner workings of bureaucracy are generally opaque, but for reliable administration this is not necessarily a bad thing. Black boxes are a mainstay in computing because they hide inner complexity and also protect inner details from outside corruption.

As Alva Couch and his students have pointed out, the computer science black-box notion of closure gives us a level of predictability, by locking out the environment that generally confounds predictability. This is the same environment that can screw up chains of preconditions, as Alva's work has taken some pains to model in detail. The trouble is that, although closure is easily implemented for things such as database transactions, it is quite difficult to implement in the area of system administration, because systems are constantly being exposed to the environment by uncontrollable backdoors. Moreover, they often share an operational state (routing tables, databases, etc.) that breaks open closures.

The story of order-independent operations is also rather nontrivial and is based on a very low-level approach to operational semantics. With cfengine, the focus has been on this approach (some think too much so), and hence it often fails to provide higher-level expressivity, which other projects such as Luke Kanies' Puppet are trying to remedy (hopefully keeping the low-level semantics intact). Paul Anderson has long told me that he sees cfengine as a low-level language that one compiles down to. This seems sensible to me. In the meantime, together with Alva Couch, I am developing a more precise theoretical model for these low-level semantics that will eventually be incorporated into cfengine 3.

Even if a configuration is reachable without any ordering problems, there are some features of behavior that depend on the order. This has to do with the fact that creation and deletion are catastrophic state-destroying operations that break commutativity on present-day operating systems. It is conceivable that one could build an operating system that did not have this property, but it would be quite difficult. A fair approximation would not be too hard to build, however, so we could have commuting operations and the order of procedures would be entirely irrelevant to the final state.

## The King Is Dead: Long Live the Laissez-Faire Army

Humans beings have a remarkable capacity to view the world in terms of subordination, and system administrators are no exception. You'd think we'd all done military service or were trying to establish ourselves as king or emperor by conquering fourth-world tribes of disorganized computers and sending them for Pygmalion execution lessons on how to behave in the Kingdom of the Data Center.

In the 1990s, as telephonic empires were crumbling, small-business entrepreneurship invaded this turf and took computer management in a different direction. Small furry businesses started making it up as they went along, thanks to tools such as small computers, UNIX, and ad hoc solutions of Windows and Macintosh. With an excitement for progress rather than control, mammals evolved and dinosaurs were left floundering. The

UNIX world has bothered itself little with the data models mentioned here: cfengine, Isconf, LCFG, and of course every site's home-brew scripts have been much more ad hoc in their approaches—with almost devil-may-care informality. And yet they work. Why?

In *Future Shock*, Toffler related an important insight, an insight that it is appropriate for us to relearn. His point was this: In the 1960s, as we remained scared of the looming presence of communism, it was assumed that the industrial age and the rise of technology meant a future that was mass-produced, in which everything was the same—there was no variation and no choice, just an overwhelming amount of factory produce, because the duplication of fixed pattern was marching to the tune and beat of industrial nations' sternest baton.

What Toffler realized was that better technology allows one to manage more variety, greater diversity, and, importantly, greater choice. We do not have to fear diversity. What, after all, is the point of information technology if not to manage the complex array of specially tailored blueprints? What is the reason for improving management of productivity if not to cater for the whims and desires of minorities and special interests?

The weight of bureaucratic constraints just to maintain a large information model is overwhelming. It is too slow. If you are attached to a fleet of steel balls by a cat's cradle of elastic bands, your best career choice is not that of acrobat.

## A Bearable Lightness of Being

There is a myth that, if you do not control something, the result will be chaos. There is a belief that, if you do control something, its behavior will be in accordance with your wishes.

I believe that there is some linguistic confusion at the heart of this debate. The word we want is not "control," because that is a word of hubris. You can tame a horse but you will never control it. There is a world of difference between control and management. Toffler pointed out the answers in 1970. We are fighting the wrong battles.

> Rising novelty renders irrelevant the traditional goals of our chief institutions. . . . Acceleration produces a faster turnover of goals. Diversity or fragmentation leads to a relentless multiplication of goals. Caught in this churning, goal-cluttered environment, we stagger, future shocked, from crisis to crisis, pursuing a welter of conflicting and self-cancelling purposes.

The real measure of intellectual achievement is to take something complex and make it simpler—by suitable abstraction. Anyone can make excruciating syntax, an exhaustive list, or a database of every possible detail. There is absolutely no evidence that tight bureaucracy leads to greater predictability. What can lead to predictability is clearer semantics—perhaps with a lighter touch.

In the next episode, I want to dispel a related myth: why centralization is not the necessity that has generally been implied.

## REFERENCES

[1] *Future Shock*, A. Toffler (Random House, 1970).

[2] "On the Theory of System Administration," M. Burgess, *Science of Computer Programming* 49, 1–46, 2003.

DAVE JOSEPHSEN

# homeless vikings

## SHORT-LIVED BGP SESSION HIJACKING—A NEW CHAPTER IN THE SPAM WARS

Dave Josephsen is author of the upcoming book *Building Monitoring Infrastructure with Nagios* (Addison-Wesley). He currently works as the senior systems administrator for a small Web hosting company and donates his spare time to the SourceMage GNU Linux project.

*dave-usenix@skeptech.org*

**THE FIRST UNSOLICITED, COMMER-** cially motivated bulk email was sent on ARPANET in 1978 by a DEC representative named Gary Thuerk [1]. A full 28 years later, spam has evolved into a 55-billion-messages-per-day [2] global epidemic that has affected areas of technology unimaginable by the ARPANET engineers of 1978. This article will chronicle the history of the spam wars, a war that has almost always been waged along two technological fronts: those of content filtering and delivery countermeasures. By examining the history of the arms race in the context of recent attacks with zombied PCs and short-lived BGP session hijacks, I conclude that one of these fronts may in fact be a dead end and worth abandoning altogether.

From 1978 to 1994, the business of spam remained a nonissue because email itself was in an infantile state. In the early 1990s, most spam was sent in the context of USENET newsgroups, and by a few identifiable individuals, such as Canter, Siegel, and Wolff [3]. In 1994 the Net witnessed its first real spam, sometimes referred to as the "spam heard round the world," when Canter and Siegel's "green card" message was sent to at least 6000 USENET groups [4].

In the early days, retribution was swift [5], but things degenerated quickly. In 1995, Floodgate, the first commercially available spamware, was available. By 1996, four more automated spam packages were available for sale, as were lists of millions of email addresses [6]. The spammers wasted no time legitimizing their so-called business model with various pro-spam trade groups such as the Freedom Knights and the IEMMC and proceeded to reach millions of USENET subscribers with their marketing messages. The lure of free marketing combined with the lack of protocol security set the stage for the inevitable war that rages on to this day.

Almost immediately, two technological methodologies appeared to combat spam. The first type focused on the content of the message in question, and the second type on indicators such as the email or IP address of the sender. These divergent paths have evolved largely independent of each other as the spam attacks have become more

frequent and increasingly complex. Content filters eventually moved toward statistical learning, whereas delivery countermeasures evolved increasingly sophisticated challenge/response mechanisms. Let's examine the lineage of each front independently, beginning with content filters.

## Content Filters

The earliest example of automated content filtering might be Nancy McGough's procmail filtering FAQ, published in 1994 and still available at http://www.faqs.org/faqs/mail/filtering-faq/. Early spam messages were very much singular events [7]. The defenders of the time knew who would be sending spam, and sometimes even when, so early content filters needed to do little more than look for static strings in the message body.

Static string searching continued to work well for many people until around the year 2000, when various content obfuscation techniques became prevalent in spam [6]. The word obfuscation games continued for a number of years, with spammers using misspellings, character spacing. and a multitude of other HTML- and MIME-based [8] techniques to bypass word filters. For a while the defenders followed in step, adding html parsers and character transformation algorithms to their content filters.

In late 2000 and early 2001, based on an idea from Paul Vixie, an innovative content filter was created which worked by taking a fuzzy checksum of a message and comparing it to a database of known spam checksums. Two implementations of this idea exist today in The Distributed Checksum Clearinghouse (http://www.rhyolite.com/anti-spam/dcc/) and Vipul's Razor (http://razor.sourceforge.net/). Spammers responded with attempts to poison the checksum database by reporting legitimate messages to the abuse lists and by adding unique gibberish to the messages in an effort to make the checksums more "different."

Then, in August 2002, Paul Graham published his seminal paper, "A Plan for Spam" [9], in which he publicly made the case for Bayesian learning algorithms for spam classification. Prior work existed [10, 11], but ironically Graham's less mathematically rigorous approach made the technique far more effective [12]. At least a dozen Bayesian implementations exist today.

Since 2002, several improvements have been made to Graham's core idea; these include the addition of several classification algorithms, such as Robinson's inverse chi-square [13], and data purification techniques, such as Bayesian noise reduction [14]. But overall, naive Bayesian classifiers have been unanswered by the spammers for four years and are therefore considered a category killer for content-based filters. Where researchers have had success against Bayesian filters, it has been by training other Bayesian filters to use against them [15].

## Delivery Countermeasures

On the delivery countermeasure front, examples of blacklists date back to November 1994 [3], when USENET "remove lists," consisting of malicious sender addresses, were used to remove unwanted messages. In those early days, reporting abuse to a spammer's ISP yielded swift results [5]. By 1995 the USENET community's outrage over spam abuse had outweighed its censorship fears, and UDPs (USENET Death Penalties) were used to block all posts from malicious sites. At the time, the sites weren't necessarily considered malicious; the UDP was most often used to "persuade" the admin-

istrators of a particular site to take care of its abuse problems after more diplomatic means had failed [16].

In non-USENET circles, sender address-based filtering was becoming more and more common, forcing spammers toward joe-job attacks. By 1996, the abuse reports and sender filtering resulted in the spammers' use of relay systems to deliver their mail. In 1997 the term "open relay" was coined to describe a mail server that would relay mail for any recipient from any sender. The first real-time spam blacklists (RBLs) appeared the same year [6].

From 1998 to 2002, so many delivery countermeasures had been proposed and beaten that they are too numerous to mention. Attempts at using blacklists were thwarted by relays, whitelists were met by directory harvest attacks and more joe-job spoofing, and greylists still showed promise but wreaked havoc with noncompliant MTAs. A few payment-based systems were proposed in this period, including micropayment-based "e-stamps" [17] and the CPU-based idea that eventually became hashcash [18]. These are not particularly effective against spoofing attacks and never gained widespread adoption. Direct challenge/response systems were ineffective, owing to forged "from" headers, and were generally considered rude. Most of the other solutions of the time were in one way or another thwarted by spammers simply adopting someone else's net identity, through open relays, or with header spoofing, or using a combination of the two.

The ease with which the spammers abused valid credentials was clearly frustrating to those designing delivery countermeasure systems. Many in this period became convinced that the problem with SMTP was the lack of sender authentication. In June of 2002, Paul Vixie wrote a paper entitled "Repudiating MAIL FROM" [19] which became the basis for SPF, or Sender Policy Framework. SPF is a DNS-based authentication mechanism which calls for the use of MX-like DNS records for mail servers that send mail, instead of those that receive them.

Although SPF was in clear violation of the SMTP RFCs and broke important functionality such as forwarding, many (especially in the business community) lauded SPF as the silver bullet that would once and for all solve the spam problem, especially when it was embraced by Microsoft and AOL. But, alas, SPF too has met with defeat [20] (although many vendors still encourage its use).

The year 2000 witnessed the first large-scale distributed denial-of-service attack against multiple prominent Web sites, including Yahoo! and eBay. The attack, launched by a Canadian teenager, brought public attention to the problem of botnets. A recent IronPort study found that 80% of the spam currently sent on the Internet is sent through similar collections of zombied PCs [2]. In one way or another, zombies make moot most of the remaining challenge/response systems of today. By directly making use of "grandma's" PC to send their message, spammers are nearly assured of success in a challenge/response scenario.

Today, RBLs remain the largest and most widely used tool on the delivery countermeasures front, despite the questionable ethics and legal entanglements of the RBL managers themselves [21, 22, 23]. The ease of setup, combined with a quantifiable reduction in spam, makes RBLs a popular choice with system administrators looking for a quick fix so that they can get back to their "real" work.

## BGP Prefix Hijacks

However, a recent paper by Anirudh Ramachandran and Nick Feamster [24] may change all that by providing a sneak peek at the next battlefield on the delivery countermeasures front. The Feamster paper provides the first documented evidence of spammers using short-lived BGP prefix hijacks against RBLs to get their mail delivered. Since you may not be familiar with the technique, I'll briefly summarize.

Prefix hijacking can happen a couple of different ways. In the first scenario, the hijacker advertises a huge netblock, for example, 12.0.0.0/8. Much of the space in this netblock is unallocated, or allocated but unused. More specific advertisements in this netblock will take precedence over larger ones, so in practice, the attacker won't interrupt legitimate traffic. For example, a legitimate company advertising 12.10.214.0/24 will not be affected by the hijacker's less specific advertisement.

The second scenario is more of a direct prefix hijack, whereby the hijacker advertises a legitimate netblock (yours, for example), and routers closer to the hijacker who don't or can't filter bogus announcements from their BGP peers simply believe the hijacker. This is less common in practice right now, because this sort of thing is easier to spot and has less of a payoff; some routers still believe the legitimate Autonomous System.

Prefix hijacking has been used in the past by profiteers who would combine bogus BGP announcements with RIR social engineering to take control of blocks of IP space they did not own. They would then sell these bogus netblocks to unwitting organizations. In the past few years, however, the network engineering and security communities have become aware of a different kind of prefix hijack. These hijacks are very short-lived, lasting 15 minutes or less.

Why would someone hijack a route for such a short amount of time? For the readership of this magazine, it's probably not a huge test of the imagination. In fact, pretty much any illicit behavior you happen to fancy would benefit from the technique, because using addresses nobody owns makes you harder to track. If you wanted to nmap the NSA, DoS the RIAA, P2P MP3s, or perform whatever other acronyms might get you in trouble, and you wanted to do it in a quasi-untraceable manner, this might be for you. Spamming people is of course a behavior generally assumed to be associated with short-lived prefix hijacks, but while speculation abounds, very little in the way of actual evidence has been available until the Feamster paper.

Prefix hijacking attacks directly target countermeasures such as RBLs by using netblocks nobody has seen yet. It's simply a new take on the same old trick of using someone else's credentials to deliver unwanted mail. Prefix hijacks can also target SPF by making it so that large portions of the Internet might actually send their SPF DNS authentication requests right to the spammers. The bottom line is that if your anti-spam solution depends on IP addresses, you lose.

## Conclusions

I believe prefix hijacking may prove to be the proverbial nail in RBL's coffin, but even as you read this the arms race escalates on the delivery countermeasures front. RBLs for their part are evolving lower into the networking stack and I'm quite sure that this is not a good thing.

For example, the MAPS RBL now offers a BGP feed that your Cisco router can consume [25]. Given the history of the war thus far, I am skeptical that further forays toward filtering spam using incidental indicators such as IP address are going to be effective without incurring additional collateral damage. Finally, given that NBCs (naïve Bayesian classifiers) remain an effective and unanswered weapon in the fight, I find it curious that there are two fronts at all.

## REFERENCES


[1] http://www.templetons.com/brad/spamreact.html.

[2] http://www.ironport.com/company/ironport_pr_2006-06-28.html.

[3] http://www-128.ibm.com/developerworks/linux/library/l-spam/l-spam.html.

[4] http://en.wikipedia.org/wiki/Canter_&_Siegel.

[5] http://catless.ncl.ac.uk/Risks/15.79.html#subj12.

[6] http://keithlynch.net/spamline.html.

[7] http://www.l-ware.com/wsj_cybersell.htm.

[8] http://www.jgc.org/tsc/.

[9] http://www.paulgraham.com/spam.html.

[10] http://citeseer.ist.psu.edu/sahami98bayesian.html.

[11] http://citeseer.ist.psu.edu/pantel98spamcop.html.

[12] http://www.paulgraham.com/better.html.

[13] http://radio.weblogs.com/0101454/stories/2002/09/16/spamDetection.html.

[14] For example, http://freshmeat.net/projects/libbnr/.

[15] http://www.jgc.org/SpamConference011604.pps.

[16] http://www.stopspam.org/faqs/udp.html.

[17] http://www.templetons.com/brad/spam/estamps.html.

[18] http://www.hashcash.org/papers/hashcash.pdf.

[19] http://sa.vix.com/~vixie/mailfrom.txt.

[20] http://www.theregister.co.uk/2004/09/03/email_authentication_spam/.

[21] http://www.internetnews.com/dev-news/article.php/10_995251.

[22] http://csifdocs.cs.ucdavis.edu/tiki-download_wiki_attachment.php?attId=431.

[23] http://www.peacefire.org/stealth/group-statement.5-17-2001.html.

[24] http://www-static.cc.gatech.edu/~feamster/publications/p396-ramachandran.pdf.

[25] http://www.pch.net/documents/tutorials/maps-rbl-bgp-cisco-config-faq.html.

DAVID BLANK-EDELMAN

# practical Perl tools: give me my woobie back

David N. Blank-Edelman is the Director of Technology at the Northeastern University College of Computer and Information Science and the author of the book *Perl for System Administration* (O'Reilly, 2000). He has spent the past 20 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 is one of the LISA '06 Invited Talks co-chairs.

*dnb@ccs.neu.edu*

**EVERYONE I KNOW IN OUR FIELD** who has been working with security for any reasonable length of time is walking around these days with their cynicism polished to a mirror sheen. I'm guessing part of this comes from the amount of Keystone Kopp–ish activity taking place every day in the name of "security." I know my eyes roll so much in response to the rules du jour at an airport that I'm starting to look like Cookie Monster.

So in that spirit, let's see what sort of "security theatre" we can perform for others with our Perl code. I want to posit the following question: What can I do to make you feel more secure about running my Perl code?

We're going to look at a set of surface-level changes you can make so that others feel better. They may actually help make the code more secure; they may not. But after all, this is the-a-tre! The notion here is we're trying to reduce fear in the user as she or he runs the code. To that end, here are five ways to address five different kinds of anxiety.

## 1. When Will It All End? (Fear of Infinite Run-Time)

For anything larger than a trivial script that completes quickly, users want to have a sense of where the code is in its process and how long it will take to complete. There's one reason why installer programs always feature some sort of progress thermometer. Humans are willing to suffer all kinds of pain, including excruciating ennui, if they are constantly reassured that it will pass, and they have some sense as to when that will happen.

There are a number of modules that can make displaying the progress of a process easy. Let's look at two of them. The first is fairly standard. Term::ProgressBar can show a standard progress thermometer:

```
use Term::ProgressBar;

my $endvalue = 500;
my $pbar = Term::ProgressBar->new($endvalue);

foreach my $value (0..$endvalue){
  # do something profound here instead of sleep.
  # (actually, for a new parent, that is pretty
  # profound...)
  sleep 1;
        $pbar->update($value);
}
```

This will produce a lovely thermometer that looks something like this:

```
39% [=========                                   ]
```

Term::ProgressBar has a number of features worth reading the documentation to discover. Especially impressive is its ability to tell you the most efficient way to call its routines. Though we didn't make use of this information in the previous example, update() actually returns the next value that's required to cause the display to change. To see how this might work, imagine a process that has 100,000 steps. Calling update() at the tenth step doesn't do anything because that value isn't large enough to move the thermometer over another notch. It's a wasted subroutine call. If we store the return value of each update() call, we can choose to only call update() again when it matters. This can be a big efficiency win for larger jobs.

A second module worth mentioning came up in the April 2006 column. In that column I demonstrated the use of Smart::Comments. This magic module will actually take specially formatted comments in your code and make them come alive. The example I gave in April was this:

```
use Smart::Comments;

for $i (0 .. 100) { ### Cogitating |===[%]    |
        think_about($i);
}

sub think_about {
        sleep 1; # deep ponder
}
```

The result is a cool progress bar that looks like this as the program runs:

```
Cogitating |[2%]                    |
Cogitating |====[37%]                |   (about 1 minute remaining)
Cogitating |=============[71%]       |   (about 30 seconds remaining)
Cogitating |==========================|
```

## 2. Hello? Tap . . . Tap . . . Is This Thing On? (Fear That the Script Has Locked Up)

Sometimes your code performs a task of indeterminate length or with an unknowable number of steps before completion. We can't use modules such as Term::ProgressBar in those cases because we have no idea what the end value will be; ten steps could represent 10% completion or .00010% completion. Even though we can't put up a pretty thermometer in cases like this, it is still important to relieve those running the program of their anxiety that things might not actually be progressing.

A simple 'print "Working..."' at the beginning of the process just doesn't cut it. The next best thing (and it's not particularly good) is something like this:

```
while (do_something){ print ".";}
```

This sort of thing works fine for small numbers of operations but it leads to an indistinguishable blizzard of periods marching across the screen when the number of steps is anything but a small amount. Yes, something like this could be used:

```
my $counter;
while (do_something){
        print "." if ($counter++ % 100 == 0); # print every 100 steps
}
```

but we can do better.

I'm old enough to wax nostalgic about Sun2 machine boot sequences, so I have a soft spot in my heart for Term::Twiddle. This module provides a spinner like the one you see in either the Sun or FreeBSD boot process. That's the little animated cursor that prints the following characters in sequence in the same spot on the screen so it appears to spin:

```
\ | / -
```

(For those of you who want to play along with the home-game version, feel free to cut out the previous line, paste each character on its own card, and make a flip-book.)

Term::Twiddle has many customization options available, but my personal favorite is the following (quoted from the documentation):

> probability . . . The purpose of this is to create a random rate of change for the thingy, giving the impression that whatever the user is waiting for is certainly doing a lot of work (e.g., as the rate slows, the computer is working harder, as the rate increases, the computer is working very fast. Either way your computer looks good!).

Using Term::Twiddle is easy:

```
use Term::Twiddle;
my $spinner = new Term::Twiddle;
$spinner->start; # start the spinner a'spinnin'
# do_something code
$spinner->stop;  # fin
```

The one gotcha worth noting for Term::Twiddle is that the do_something section in the this example can't include any sleep() calls, since sleep() potentially messes with the interval timers that allow the module to work.

If you don't like that restriction, you may be interested in another module in that family, Term::Activity, which provides a slightly less disingenuous view of how things are progressing. Term::Twiddle sets off this animated cursor thingy that changes without any direct connection to the actual process it is purporting to show working. In contrast, Term::Activity actually requires your code to call a tick() subroutine every time it wants another step in the process to be registered (and the display to change). The code then looks like this:

```
use Term::Activity;
my $progress = new Term::Activity;
while (things_are_happening){
        # do_something code
        $progress->tick();
}
```

The result is an ASCII wavelike thingy (install the module to see what I mean) that also counts the number of times tick() has been called and the interval between tick()s. All of this increases the warm fuzzy count of the person running the code.

## 3. Perl the Destroyer (Fear That the Script Could Be Doing Damage)

If I take a BB gun, aim it at my foot, and pull the trigger I can roughly approximate the process many Perl neophytes go through when they first learn how to write filesystem walking/changing code. Modules such as File::Find or the even spiffier File::Find::Rule make it super easy to write code that will nuke large chunks of your filesystem with very little effort. For this reason it is crucial that your code make it fairly hard to take destructive actions. If your code mass-removes files by default, someone who stumbles on your "cleanup.pl" script will be in a for a rude surprise when he or she decides to run it to see how it works. Granted, this is perhaps the last time that person will do something so ill-advised, but still, this isn't exactly the best pedagogical technique.

One easy way to avoid this situation is to force the user to call the script with a large and slightly unwieldy command-line switch if deletions are really desired. Something such as --deleteFilesAtWill could be used. A quick warning is called for here: If you are using a module such as Getopt::Long that handles abbreviated switches automatically and by default, be sure to pick something that doesn't abbreviate to a common switch name. This means that arguments such as --debugByDeletion, --verboseMakeGoByeBye, or --helpMeNukeMyFilesystem are probably right out.

## 4. Check What Condition Your Condition Is In (Fear That the Script Will Run for a Long Time and Then Fail)

As tempted as I am, I'm not going to harp more on the test-first methodology we discussed back in the April column. Instead let me harp on a variation of that idea. It can inspire confidence if your program can run a brief self-test before it runs. This test can check necessary conditions the program needs before running. For example:

- Is the database it needs hot and ready to go?
- Are the file permissions on key configuration files still ok?
- Do the working directories used by the program exist, and are they writeable?
- Is DNS reverse-lookup working fine?
- Are the TLS/SSL certificates that are going to be used still valid?

Code like this:

```
use Test::Simple tests => 5;
ok(configs_owned_by_user(), 'config files are fine');
ok(-w $tempdir, '$tempdir ready for writing');
ok(test_database(), 'database ready to go');
ok(reverse_lookup('192.168.0.1') eq 'router.example.com', 'DNS ok');
ok(check_certs($certdir/$certname),'TLS cert is valid');
```

produces comforting output like this when everything is going smoothly:

```
1..5
ok 1 - config files are fine
ok 2 - /var/tmp ready for writing
ok 3 - database ready to go
ok 4 - DNS ok
ok 5 - TLS cert is valid
```

so you know Thunderbirds are Go!

Here's a tip: If the program itself is due to have a long runtime there's a little more leeway for how long this self-test should take, but beware of letting it drag on too long. At a certain point it becomes like a folk singer who spends more time tuning the guitar than playing it. Eventually the audience revolts and starts throwing berets and bongos.

## 5. Breakfast. Lunch. I Said Lunch, Not Launch! (Fear of Using the Script Incorrectly)

For our last anxiety-reducing tip of this issue we're going to look at an easy way to embed the documentation for a script in that script. This method will allow the documentation to travel with the script (versus a separate manual page) without getting in the way of the code itself.

Pod::Usage makes it easy to provide two types of documentation on demand: the standard short "USAGE" message for a summary of script purpose and options or a full-blown manual page. It can actually produce something in between, but in practice I only use it for these two.

Here's how it works: Code outside of Pod::Usage is responsible for the parsing of the program options that choose how to call Pod::Usage. In practice, this means the usual option-parsing code that probably looks something like this:

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

# I prefer to store the options I receive in a hash
my %options;
GetOptions(\%options, 'help', 'man', {more options...});
```

Now we dispatch based on the switches the script receives:

```
# handle help or man page request
pod2usage(-exitstatus => 0, -verbose=> 0) if (exists $options{help});
pod2usage(-exitstatus => 0, -verbose=> 2) if (exists $options{man});
```

In this case we're calling Pod::Usage with two parameters:

1. Exit status: When Pod::Usage exits after doing its job, what should the exit value of script be as a whole? In the preceding code, we say it should be 0 (i.e., success), since the script will have successfully done its job of printing out documentation upon request. In some cases (e.g., if the script detects that the user hasn't called an option with the right arguments) we'll want to set this exit status to indicate failure. That way the program can exit with an error message and set the status accordingly. For example:

```
# abort if we don't get some key info about our ice cream cone
die "--flavor <name> not specified, aborting... (try --man)\n"
        unless (exists $options{flavor});
```

2. Verbosity: How verbose (i.e., USAGE message only or full manual page) should Pod::Usage be? Verbose level 0 prints the former; level 2 prints the latter.

Are we done? Well, almost. All we have to do now is arrange for there to be some documentation to display. (You did write documentation in parallel with the code, right?) The documentation typically lives at the bottom of the script in POD format (see "perldoc perlpod" and "perldoc perlsyn" for more info) after an __END__ token:

```
# lovely script here above this point in the file ...

__END__

=head1 NAME

    makecone - construct an ice cream cone

=head1 SYNOPSIS

    makecone [options]

        Options:
            --flavor <name of flavor>   specify flavor for ice cream (required)
            --help                      print usage message only
            --man                       show entire man page for this script

=head1 DESCRIPTION
```

. . . and so on.

Now if a user calls your script with --help or --man he or she will receive enough documentation to gain some sense of whether your script is being used correctly.

And with those warm cockles, I'm afraid it is time to end this issue's column. Take care, and I'll see you next time.

ROBERT HASKINS

# ISPadmin: wireless

Robert Haskins has been a UNIX system administrator since graduating from the University of Maine with a B.A. in computer science. Robert is employed by Shentel, a fast-growing network services provider based in Edinburg, Virginia. He is lead author of *Slamming Spam: A Guide for System Administrators* (Addison-Wesley, 2005).

*rhaskins@usenix.org*

**THIS ARTICLE FOCUSES ON TECH-** nologies and hardware that service providers would typically use for deploying wireless networks. Nonlicensed spectrum products are the focus here, since licensed spectrum products are usually beyond the budget of a typical ISP. That doesn't mean licensed products are not used in the service provider market; it just means that licensed spectrum usage is a much smaller segment of that market than is unlicensed spectrum usage.

This article is not meant to be a primer on wireless technologies. The references contain a few pointers to resources where the reader can obtain additional information on wireless technologies [1, 2]. Note that this article is based upon U.S. standards, so readers in other countries should consult their local regulatory agency's rules regarding what is licensed in their particular country.

(Disclaimer: My employer uses many of the products used in this article, including Colubris and Motorola.)

## Background

The three applications and frequency ranges typically used (in parentheses) in service provider networks can be broken down as follows:

- Last mile (900 MHz)
- Hotspot, last mile (2.4 GHz)
- Backhaul, hotspot, and last mile (5.x GHz)

In the United States, most of this space is unlicensed spectrum, allocated by the Federal Communications Commission (FCC). One of the major differences between unlicensed and licensed spectrum is that with unlicensed spectrum, the operator of the equipment is required to resolve interference issues. With licensed products, there is no spectrum "sharing," so interference is usually minimal.

One unexpected application of a wireless provider network is in the area of device tracking in metropolitan areas. Often the traditional satellite GPS signal is unusable where there is a lot of interference, such as near a tall building. By triangulating multiple wireless device signals from an ISP's existing wireless network, the location of the device can be determined. Although this will

never replace GPS altogether, it is useful for some metropolitan regions and applications.

## Last Mile

Last mile access can be thought of as a DSL or cable modem access replacement. That is, the service provider's POPs connect to the end customer via wireless connection. This "last mile" access eliminates the need for using a local exchange carrier's (LEC's) network, saving cost and possibly time to deployment. One useful application of wireless is allowing access for people living in remote areas whose network access is nonexistent or not more than 128 kbps.

The frequency behind many last mile products is 900 MHz, though 2.4 and 5.x GHz can also be used. As with many wireless technologies, line of site is important but not necessary. Many 900-MHz products are able to get six miles of non-line-of-sight service at 3 Mbps [3]. Typical pricing for the Trango solution is $539 for the customer premises equipment (CPE) and $1595 for the radio and antenna, which can serve 126 CPEs. Motorola Canopy [4] is another offering in this space.

Interference with 900 MHz can be caused by several things: other ISPs running 900 MHz equipment, older wireless phones, scanners, baby monitors, and video senders. Mitigation of the interference is the responsibility of the equipment owner (i.e., the service provider) when unlicensed products are used.

## Hotspot

The first widely deployed hotspot IEEE standard was 802.11b, which used the 2.4-GHz frequency at up to 11 MB/s (a maximum of 300 feet from access point to subscriber). Subsequently, the 802.11a standard was developed; it used the 5.x-GHz frequency up to 54 MB/s but was not compatible with 802.11b, because it used a different set of frequencies. Finally, the 802.1g standard allows the 2.4-GHz range up to 54 MB/s and retains compatibility with IEEE 802.11b.

One benefit of deploying 802.11 hotspots (Wi-Fi) is that many portable devices (e.g., laptops) contain embedded 802.11 access devices. This eliminates the need for the provider to ship a CPE device to the subscriber. (Of course, it also eliminates a potential one-time revenue source, but service providers aren't usually focused on selling hardware.)

Hotspots can be deployed using "consumer grade" access points such as Cisco/Linksys or Netgear. However, these devices are often lacking needed features in a service provider's network. The list includes:

- Remote manageability/monitoring (SNMP)
- RADIUS authentication for end-subscriber access
- Wall/ceiling/outdoor enclosure
- Multiple wireless networks (SSIDs) within the same device

One option is for a service provider to build its own access points. Many open source solutions exist for manufacturing your own hardware (see the October 2005 edition of ISPadmin on the topic of embedded systems). However, the cost and trouble involved in a home-grown solution usually makes a commercial solution more attractive. An option for a low-priced gateway would be NoCatAuth [6]. This open source project handles authentication on low-cost hardware.

Along with the radio, a provider will want to control some activities of the subscriber by deploying a gateway that can provide a login screen, authenticate the subscriber, and handle redirection and other similar features. This gateway can be combined with the radio (e.g., the Colubris 3300R) or can be separate (the Colubris MSC-5200/5500 or Nomadix AG3000 [5]). In large deployments, multiple radios can be attached to the 5000-series device, lowering the cost of the project. One manufacturer of service-provider–grade access points is Colubris.

The Colubris 3300R series [7] devices are a good example of what a service provider might use in a combined gateway and radio device. In addition to the necessary but often omitted features in the bulleted list on p. 72, the 3300R series includes, but is not limited to, the following features:

- Multiple radios
- DNS relay and SMTP redirection
- 100 subscriber maximum
- Quality of Service (QoS) management

## Backhaul

Backhaul involves moving data between different points on the provider's network. For example, the provider might aggregate all customer traffic at two points in its network. From those two points, the service provider would purchase two connections to the Internet. So the question comes down to how to get all the subscriber traffic to one or both of those Internet connections. This is known as backhaul.

Back in the days before unlicensed wireless, the only option for backhaul was either an LEC-provided data circuit (T1, DS3, etc.) or an expensive license spectrum (RF or microwave-based equipment). The licensed-spectrum solution was not typically an option for an undercapitalized ISP. However, once the nonlicensed spectrum was open and products were available, wireless and the ability to bypass the LEC became a viable option. Also, the ability to reach remote non–LEC-served areas was possible with low-cost wireless backhaul.

Wireless backhaul equipment is normally very similar to the "last mile" equipment, but with an antenna that sends a signal in a narrow range (as opposed to the omnidirectional antenna typically used in last mile applications). Backhaul via wireless is not without the usual issues, however. Some of these issues include:

- Interference with other providers
- Interference with wireless devices
- Line of site
- Power

The Trango Atlas 5010-EXT [8] offers 45 Mbps of bridged Ethernet up to 20 miles at $2795 per connection. This is quite cost-effective when compared to the recurring cost of a DS3 circuit. Another option would be the Motorola Canopy 5430BH product, which offers 60 Mbps line-of-sight to 124 miles.

## Troubleshooting

No article on wireless would be complete without covering issues related to deploying wireless networks. A wireless spectrum analyzer is extremely helpful in troubleshooting interference with other devices and networks. Like traditional network analyzers, they come in two types: standalone

dedicated devices, and software (and possibly hardware) that runs on another device such as a laptop or handheld PC. The references list a couple of devices, both dedicated [9] and portable-PC-based [10]. Another tool that is useful for troubleshooting hotspots is a software program such as Netstumbler [11]. This is an application that runs on a Wi-Fi–enabled laptop and tells you what networks are within the range of the wireless radio.

When designing and deploying a wireless network, be sure to check overall throughput and make sure latency doesn't affect performance. Latency can be high if there are a large number of hops or if interference is present. Also, don't deploy wireless networks during the winter when foliage is missing and expect it to work during the summer when everything is in full bloom!

## WiMAX

WiMAX (IEEE standards 802.16d and 802.16e) is a next-generation wireless protocol supporting a wide frequency range (2–66 GHz) and fast speeds. It is designed as a "last mile" replacement, offering multi-megabit speeds, with a range of 1–5 miles without line of sight. It may be useful for backhaul applications, but use in this area is currently somewhat limited. The standard could conceivably even replace the 802.11a/b/g wireless "hotspot," though total replacement is unlikely at least over the next few years, owing to the large installed base of 802.11 devices and networks.

WiMAX can be used with either licensed or unlicensed spectrum, making it a flexible choice for spectrum license holders and nonlicense holders as well. It can be deployed as a fixed or mobile (handheld/laptop) configuration. Although it is often hard to distinguish between the hype and news, references [12] and [13] give some good background on the WiMAX arena.

## Community Access

Because of the low cost and lack of LEC involvement in deployments, wireless technology is an excellent way for communities to "build their own ISP." In fact, several U.S. cities are sponsoring wireless access deployments within their boundaries. Many other entities are getting into the act as well; sponsors can range from counties and homeowner associations to technologically oriented geeks. Because they operate like a nonprofit, the cost can be less than traditional access methods (DSL and cable modems) and built to serve underutilized and/or remote or rural areas. However, just because these organizations are nonprofit doesn't mean that money-paying subscribers will accept poor service!

Mesh networking [14] is a methodology used by some community-supported enthusiasts (and others) to reduce backhaul costs. Instead of routing all traffic directly to the Internet connections on the network, mesh networks route traffic through several peer (subscriber) nodes prior to hitting a node that is Internet-connected. This has the benefit of potentially reduced cost, depending on how the network is architected. The downside is that it takes specialized software to handle routing (and, potentially, billing) and other tasks that are not widely supported in current consumer firewall/router/access points.

I'd like to thank Bob Alexander of Shentel for his assistance with this article.

## REFERENCES


[1] http://www.practicallynetworked.com/pg/wireless_networking_bkgrounder.htm.

[2] *802.11 Wireless Networks: The Definitive Guide*, 2nd ed., Matthew Gast (O'Reilly Media, 2005).

[3] Trango M900S: http://www.trangobroadband.com/products/m900s.shtml.

[4] Motorola Canopy: http://motorola.canopywireless.com/products/specshome.php.

[5] Nomadix AG3000: http://www.nomadix.com/products/platforms/ag3000/.

[6] NoCatAuth: http://nocat.net/.

[7] Colubris 3300R: http://www.colubris.com/downloads/datasheets/DS_MSC_3000.pdf.

[8] Trango Atlas 5010: http://www.trangobroadband.com/products/atlas_5010.shtml?id=bb.

[9] BK Precision Handheld 8.5-GHz Spectrum Analyzer Model #2658: http://www.bkprecision.com/www/np_specs.asp?m=2650.

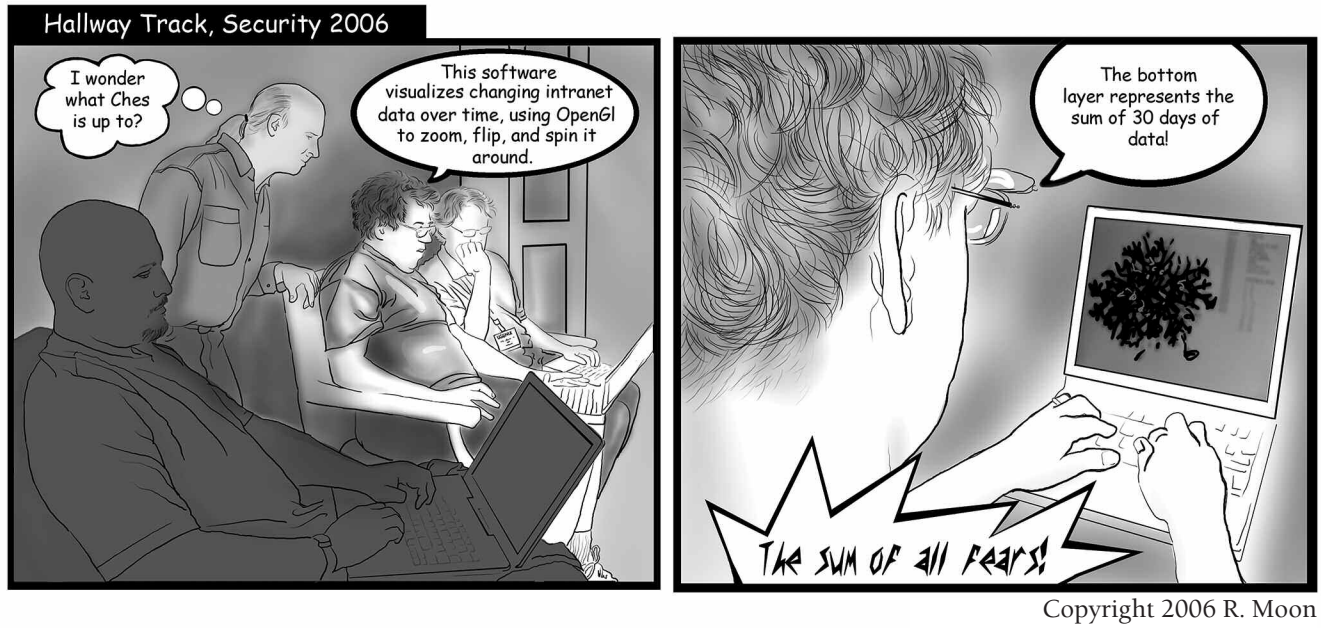
[10] Fluke Networks AnalyzeAir Wi-Fi Spectrum Analyzer: http://www.flukenetworks.com/fnet/en-us/products/AnalyzeAir/MOA.htm.

[11] Netstumbler: http://www.netstumbler.com/.

[12] ZD Net UK article on WiMAX: http://news.zdnet.co.uk/communications/wireless/0,39020348,39241047,00.htm.

[13] WiMAX Forum: http://www.wimaxforum.org/home/.

[14] Mesh networking page from Wikipedia: http://en.wikipedia.org/wiki/Mesh_network.

HEISON CHAK

# VoIP watch: security

Heison Chak is a system and network administrator at SOMA Networks. He focuses on network management and performance analysis of data and voice networks. Heison has been an active member of the Asterisk community since 2003.

*heison@chak.ca*

**AS I WAS WAITING AT TORONTO** Pearson Airport to board my flight, my early evening nap was interrupted by a familar sound—the default ring tone of a Cisco phone. Pearson is one of the airports that have taken the step to convert most (if not all) telephone communication to VoIP about three years ago. Today, Cisco IP handsets can be seen just about everywhere throughout the airport, from airline counters to information kiosks.

Although most VoIP implementations focus on voice quality, latency, and interoperability, the first question that comes to my mind is, How is security handled at this scale of deployment? In other words, how are confidentiality, availability, and integrity issues being addressed?

These state-of-the-art telephony systems promise to cut communication costs by carrying more voice calls than traditional switched circuit networks and enable enhanced services such as unified communications. However, as with traditional telephony, vulnerability to theft of service, denial of service attacks, and eavesdropping are all concerns for organizations deploying VoIP, and the consequences can be far more serious.

## Confidentiality

As with data networks, VoIP security needs to be handled in a similar context, which may involve properly locking down servers and placing them behind firewalls, patching against vulnerabilities, and monitoring activities with intrusion-detection systems. Call detail records contain identity of callers and call patterns and should be treated with the same level of sensitivity as the actual content of a communication channel. Since voice travels in packets over IP networks, hackers can use data-sniffing and other hacking tools to carry out unauthorized wiretapping. It is possible to identify, modify, and play back voice traffic traversing such networks. For example, the vomit utility converts a conversation of a Cisco IP phone in G.711 (a codec) to a wave file that can be played back with a sound player.

```
$ vomit -r phone.dump | waveplay -S8000 -B16 -C1
```

Break-ins of a call manager host or soft switch that is directly accessible from the Internet or an

open network on a university campus could result in loss or compromise of sensitive data. Credit card numbers, social security numbers, and other important PINs entered during a phone call may end up in the wrong hands, allowing identity theft. A compromised gateway could turn into financial damages as a result of theft of use.

## Availability

When designing VoIP networks, one should be aware that a VoIP packet stream exhibits behavior different from that of data packets. Although VoIP packets are small, they come in at a higher rate than do most data packets. Consider a regular data switch deployed in a VoIP network trying to handle tens or hundreds of VoIP devices communicating at 20-ms packetization interval (voice media separated into 20-ms frames for transmission); the switch can easily grind to a halt with high packet rates while utilization is still low. Buffers, echo canceller, and interface queues on routers and switches may also introduce additional delay, contributing to unpleasant conversations. Reducing hop count and increasing bandwidth may ease some of these delay issues. In the back office, when VoIP equipment is deployed alongside data equipment, one must size UPS and HVAC accordingly. Provisioning additional UPS power and runtime for soft switches and PoE capability will avoid an embarrassing situation should UPS power be overdrawn in a failover situation. The airflow in a small riser room may no longer be adequate for the VoIP PBX system. One of the biggest challenges in VoIP is providing telephony-like system uptime with general-purpose computer hardware and software. The discrete network elements like to advertise 4 or 5 9s of availability, but the ITU Telcordia estimates overall PSTN availability to be 99.94%. This metric also implies that 99.94% is the end-to-end requirement for VoIP to achieve PSTN equivalence.

## Integrity

Voice packets should not be altered, callerID should reflect the true identity of a caller, and call detail records should be guarded with care, so that billing reports can be generated accurately. These requirements may sound reasonable and simple, but the fact is that they may be technically difficult to achieve. With NAT (Network Address Translation) and some widely used Layer 4 protocols (e.g., SIP and H.323), Layer 3 addresses can often be found in the wrong layer, making them difficult to deal with. For example, the "contact" address of a SIP packet originated from a host with an RFC1918 address behind a NAT firewall is not reachable from the Internet. An ALG (application layer gateway) and the middlebox solution are designed to disassemble the packet and replace the Layer 4 SDP (session description protocol) contact address of a SIP packet with a routable address of the edge router, such that return packets can be routed. It is obvious that such techniques violate the integrity of these VoIP packets, and it will continue to happen as long as the dominant VoIP protocol breaks NAT. There are many workarounds, yet the permanent fix is to avoid using NAT altogether and may be to go to IPv6 or use a protocol, such as IAX, that works well with NAT.

As with callerID, which was never really trusted in the PSTN world, emerging to VoIP makes it even easier to forge. The following Asterisk dial plan demonstrates how easy it is to alter callerID information. It sends a call to a SIP provider with callerID set to "Bill G" (you may be surprised to find out how many telephone companies actually pass the callerID

onward). In the extensions.conf configuration file used by Asterisk, changing the callerID is as simple as including a couple of lines:

```
_9.,1,SetCallerID(Bill G)
_9.,n,Dial(SIP/${provider}/${EXTEN:1})
```

## Now What?

Some suggest signing, encrypting, and tunneling VoIP packets to ensure authenticity of callers, protecting all voice stream and touch-tone keystrokes, and working around the NAT problem. Since VoIP is susceptible to delay, having to sign every single packet and crypto overhead may introduce further delay to a VoIP packet in transit. Tunneling can also impact throughput, as additional header is required, worsening the header versus payload ratio (especially for efficient codecs, such as G.729). Header compression can ease the pain but may require custom work.

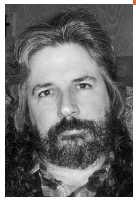
Typically, the one-way delay of a PSTN phone call is less than 150 ms. To maintain similar quality of voice over an IP network, there need to be algorithms that can perform tasks of signing and encrypting packets in a speedy fashion.

| *Delay Source (G.729)* | *On-Net Budget (ms)* |
|---|---|
| Device sample capture | 0.1 |
| Encoding delay (Alg delay + processing) | 17.5 |
| Packetization/depacketization delay | 20 |
| Move to output queue/queue delay | 0.5 |
| Access uplink transmission delay | 10 |
| Backbone network transmission delay | latency |
| Access downlink transmission delay | 10 |
| Input queue to application | 0.5 |
| Jitter buffer | 60 |
| Decoder processing delay | 2 |
| Device playout delay | 0.5 |
| Total (one-way) | 121.1+latency |

Until such algorithms become available, we may need to weigh confidentiality against usability. To match the latency in PSTN of 150 ms, when the typical VoIP latency is already 121.1 ms (ignoring network latency), any algorithm used for encryption must be so fast as to be insignificant. Perhaps we should continue to rely on our own ears to authenticate a caller's voice until standards and the required infrastructure for authentication exist.

ROBERT G. FERRELL

# /dev/random



Robert is a semiretired hacker with literary and musical pretensions who lives on a small ranch in the Texas Hill Country with his wife, five high-maintenance cats, and a studio full of drums and guitars.

*rgferrell@greatambience.com*

**KNOW WHAT I LIKE ABOUT ONLINE** auctions? Well, several things actually, but in this case I'll confine my enthusiasm to the juicy low-hanging fruit known as storage media. I'm talking "previously owned" hard disks, thumb drives, micro drives, memory sticks, compact flash cards, SM cards, and all the other variations on that theme to be found for sale to the highest bidder. Even old DLT tapes, Zip disks, and their somewhat archaic ilk can be had, for those with the equipment to read them. What is it about these utterly commonplace devices, you may well ask, that waxes my elephants? It is the treasures nestled deep in their binary bowels, of course: intact data. The number of people who yank media out of a computer or digital camera and ship it off to a never-met buyer without even bothering to delete the contents is, to choose but one from the stable of modifiers that apply, staggering. Even among those who make some attempt at sanitization, the belief that simply deleting files on a hard disk is sufficient to obliterate data is as widespread as it is erroneous. That minor misconception is what makes disk-diving entertaining.

I'm not going to cover the basics of file systems and the mechanisms for deletion of data therefrom, because I strongly suspect most anyone reading this already knows all about that sort of thing—probably more than I do, in fact. I will instead press onward and mention that the public record is liberally littered with examples of carelessly cast-off bits. Customer profiles, patient files, Privacy Act data, confidential transactions, student grades, personal communications, pirated music and video, homegrown pr0n, and just about every other embarrassing and potentially actionable manifestation of our obsession with archiving ones and zeroes for posterity are there for the bidding and the winning and the fondling. Garfinkel and Shelat in 2003 estimated that only 9% of used drives bought off an auction site fit into the "properly cleansed" category, while something like 17% still contained intact operating systems as well as user data. I'm not talking about

data recoverable only using expensive forensics suites or scanning tunnel microscopy, either. I mean plug it in, boot it up, read 'em, and reap.

Although I don't have figures at hand concerning other forms of removable data storage, I would expect them to suffer from a similar lack of deletion diligence. There have been marketing campaigns touting thumb drives, for example, as "disposable" storage. All storage is disposable, as far as that goes. Despite prolific coverage of "dumpster diving" as a lucrative technique for intelligence gathering, people still tend quite naively to equate disposal with oblivion. One of my best friends as a child was the son of a small-town garbage man. (Yep, that's what we called them back then. They hadn't come up with "sanitation engineer" yet.) His dad would find some truly amazing things in the trash on occasion: completely functional electronics, toys still in their original packaging, intact china, perfectly serviceable furniture, and so on. My buddy had, as a result, the single largest personal collection of plastic model parts in the known universe: closets and bedrooms and storage sheds full of them. If we were children today, I figure his dad would be bringing home operational thumb drives and flash memory cards for us. They're not as easy to build room-filling space stations out of, but we'd manage somehow. Nerds are nothing if not resourceful.

Giggling at embarrassing vacation photos someone thought they'd dispatched forever or riffling through breathy emails to an old lover are invasions of someone's privacy, admittedly, but they don't come close, damage-wise, to the ever more popular hobby of identity theft. Aye, there's the rub, or rather, jagged laceration, where unsanitized media are concerned. When your Social Security number, bank account, date of birth, and online passwords are in the hands of the profit-minded, you're in a world of hurt. There's an awful lot of mayhem that can be perpetrated in your name in this case and—here's the really dicey part—even though you're the victim here, it's up to *you* to prove you didn't do it. The "innocent until proven guilty" principle, which has already taken a terrific tossing around in recent years, goes right out the window and splatters messily on the pavement far below. Meanwhile, the bad guy has moved on to buying stuff using someone else's identity, blissfully unconcerned with the years it will take you to rebuild your besmirched reputation in the eyes of the financial sector. All because you, or more likely some company with whom you once did business, couldn't be bothered to wipe before flushing.

Here, of course, is where the whole concept gets seriously scary. No matter how careful you are with your personal data on equipment you control, there's not much you can do about the appalling lack of security at the myriad institutions with whom you've shared this information. Just stop and think how many hard drives in the world have your private data stored on them, any one of which falling into the wrong hands could shred your life quite thoroughly. On second thought, maybe it's better if you *don't* think about it too much. There's enough bad news permeating our daily existence as it stands now without adding yet another worry based on an event with a fairly low statistical probability of occurring, like a hurricane or a supervolcano eruption or an asteroid strike or avian flu or global warming or Lyme disease or . . . Sorry, my world view has been deteriorating steadily since I ran out of selective serotonin reuptake inhibitors the other day.

Returning briefly to the subject of used media, the Data Protection Gurus say that if you'll take the simple preliquidation precaution of overwriting everything with zeroes or ones you'll foil all but the most determined and well-funded illicit data recovery efforts. I go them one better, though. Before I discard old media, I overwrite everything with *twos*.*

*The "two" is stamped into the head of a twelve-pound sledgehammer.

# book reviews

ELIZABETH ZWICKY, SAM STOVER, AND RIK FARROW

## SECURITY AND USABILITY: DESIGNING SECURE SYSTEMS THAT PEOPLE CAN USE

*Lorrie Faith Cranor and Simson Garfinkel, eds.*

O'Reilly, 2005. 692 pages.
ISBN 0-596-00827-9

This is a collection of individual papers about security and usability. The collective message is "Security is hard. Usability is also hard. They are not actually in complete opposition, but combining them is really hard." Security people often get away with waving their hands and saying dismissively that making things more secure inherently makes them less usable, so there's no point expecting systems to be usable. This volume is devoted to the proposition that this is a cop-out. Some security interferes with some usability, but most secure systems have usability problems for the same reason that most usable systems have security problems; someone decided that the relevant property could be painted on at the end, but it's actually a design property. You have to actually know what you're doing, think deeply about it, and build it in.

This volume covers a wide range of topics from a wide range of perspectives. I found it pretty uneven and yawned my way through several articles, but on the whole it was enlightening. It includes some great tragicomic moments, such as Simson Garfinkel's paper on used disks, and the infamous "Why Johnny Can't Encrypt," in which 12 people tried to use PGP and only a third of them managed to sign and encrypt a message correctly within 90 minutes and three accidentally exposed the secret. There are also some great case studies where people actually ended up able to use the stuff, which tend to drive home the "usability is also a design property" point, and some good studies on passwords, which drive home the points that actual data can be useful and that text passwords remain popular for really good reasons.

This is a great book to consider before you design your next project that includes authentication and handy to have around to pull out figures to support arguments like "No, I don't think biometrics will solve that for you" and "Yes, I think we ought to run a nice, low-level format on that disk before we let it out of our hands." This is not a book to read from start to finish, at least for me; it starts slowly and doesn't really pick up steam until section three.

## PROTECT YOUR WINDOWS NETWORK: FROM PERIMETER TO DATA

*Jasper M. Johansson and Steve Riley*

Addison-Wesley, 2005. 549 pages.
ISBN 0-321-33643-7

This is a sensible, well-written guide to security from a Windows perspective. Of course, mostly what I mean by "sensible" is that the authors agree with me on almost all the subjects where I already knew my opinion, but I also mean that they have a balanced, rational tone and talk about broad issues instead of exclusively about details. They spend a lot of time encouraging the reader to think about security overall, instead of securing one thing and focusing on the demon of the day.

Although the book comes from a Windows perspective and occasionally gets into Windows specifics, it covers a lot more than just Windows issues. It talks about policies, about users, about physical security. It's a nice guide to the universe of security, even if you're not interested in Windows. If you are interested in Windows, it gives you a lot of important information not available elsewhere and helps sort out the genuinely important issues from the frantic hand-waving and the strange registry-setting obsessions.

This is by far the best, most readable Windows security book I've come across. Admittedly, my experience with the genre is not exhaustive, but it's large enough to have been exhausting; I've certainly hefted a bunch of them, opened them up, groaned in misery, and put them down again. This is not one of those; it's a fine addition to any security library that happens to be about Windows.

## WINDOWS SERVER 2003 SECURITY COOKBOOK

*Mike Danseglio and Robbie Allen*

O'Reilly, 2005. 479 pages.
ISBN 0-596-00753-1

Once you've read *Protect Your Windows Network* so that you understand what you want to do, this will help you figure out how to do it. Don't, under any circumstances, do it in the other order. This is a cookbook; *Protect Your Windows Network* is a menu planner and nutrition handbook. If you start with the cookbook and eat nothing but cookies and steak, it's not the cookbook's

fault, but you're not going to feel good.

The *Cookbook* does attempt to give you some background as to why you might want to do things and what might go wrong if you do them, but it's only really enough to help you if you have a firm background in the underlying issues. Once you have that background, it looks useful, providing command-line and scriptable solutions wherever possible. I'm definitely keeping it around for those moments when I know I need to beat something into submission but don't know how. I will be using caution, however; I note that it doesn't mention the oddities of the "cacls" utility, which doesn't propagate permissions. In Windows, you have to tell files to inherit permissions—cacls doesn't, so changing directory permissions won't change permissions on files that are supposed to inherit the directory permissions, until some future time when something else propagates the inheritance. This is a nasty trap, and anything that suggests you use cacls really ought to mention the issue.

### HOME NETWORK SECURITY SIMPLIFIED

*Jim Doherty and Neil Anderson*

Cisco Press, 2006. 199 pages.
ISBN 1-58720-163-1

This is a book designed for the security-naive but not technology-phobic Windows user—the kind of thing you'd hand to your more self-sufficient relatives to get them to deal with their own security. To fit network security into 199 pages, with lots of color pictures and white space, it simplifies pretty ruthlessly. For instance, it doesn't even mention the existence of non-Windows operating systems. If you were hoping it might apply to your relatives who are bothering you about the security of their Macintoshes, your hopes will be dashed.

That said, I think it does a pretty good job of covering the issues for its audience. They're more tolerant of WEP than I would be. (They do encourage the use of WPA if it's available, but they're willing to accept WEP.) I also have a nontechnical issue with their snooping advice; snooping is just as icky as a parenting technique as it is within a marriage. In either situation, you might consider logs and traces as an agreed-on way to maintain accountability instead of as a secret, an idea they don't mention.

On balance, however, the authors cover the important stuff in a friendly, accessible way, and they manage to be realistic about the dangers of the Internet. I might hand mine off to one of my more distant relatives (the close ones all run operating systems it doesn't believe in).

### DICTIONARY OF INFORMATION SECURITY

*Robert Slade*

Syngress, 2006. 222 pages.
ISBN 1-59749-115-2

This is billed as an essential reference tool; I'm not sure about that. It's not that it's a bad dictionary of information security. It's a perfectly good one, with definitions that are precise enough to be useful without being incomprehensibly technical. The jokes are small enough and rare enough to work as leavening, and there's a nice appendix with a list of other useful dictionaries. I am also somewhat awed that Slade has managed to respond to the one complaint about security books that I didn't expect to see ever handled: Appendix B has a plot and some character development.

Nonetheless, I've never thought "Gee, what I really need here is a dictionary of information security," and I don't expect I will anytime soon. I might possibly look something up in it to answer a question such as "Is that new, or did I just miss it somehow?" but most of my questions about information security terms are more likely to be answered by a search engine (for brand-new terms) or the Jargon File (for the history of terms). This book would be most useful for somebody just entering computer security, but if you're that person, and you're reading things you can't understand without the help of the dictionary, you're in over your head and need some deeper background.

### SOCKETS, SHELLCODE, PORTING, & CODING

*James C. Foster*

Syngress, 2005. 667 pages.
ISBN 1597490059

REVIEWED BY SAM STOVER

This is the final entry in a list of books that I've reviewed by this author. This book, as with the previous ones I've reviewed, comes with an upside and a downside. The upside is that there are a couple of chapters that have really excellent material. The downside is that there are only a couple, and the remaining chapters have been cut-and-pasted from other books.

Let's focus on the good first. Chapters 3, 4, and 5 explain BSD, Windows, and Java Sockets, respectively. There's lots of good material here, and although I'm not a fan of Java, it does provide at least a great foundation for both UNIX (BSD) and Windows sockets.

Chapter 6 is a good introduction to writing portable code, which is the name of the chapter, incidentally. From that, Chapter 7 launches into network-specific

portable coding. I found these two chapters to be the real gems in this book. My day-to-day life does not include any C programming, so working through these chapters was very fun and informative.

Chapter 13, "Writing Security Components," focuses on introducing the reader to the Component Object Model (COM) and implementing it with the Active Template Library (ATL). Once that foundation is laid, the final chapter, "Creating a Web Security Tool," gives a very fun glimpse into the intricacies the authors encountered when writing their own Web scanner. Very good stuff.

Now here's the bad: All of the other chapters exist in other books, namely *Writing Security Tools and Exploits, Buffer Overflow Attacks,* and *The Pen-Testers Open Source Toolkit.* There are 14 chapters in this book, and only half of them contain original material. If you own any of the other three books, be warned. The chapters are all in various stages of cut-and-pasting. It's hard to tell which chapters were lifted from which books, but the fact remains that there is a highly incestuous relationship among the four books.

As I've said before, each book on its own contains a wealth of information. The problem arises when you buy one of the other books in the hopes of moving deeper into the subject, only to find that it's the same material, sometimes verbatim. Regardless of the right or wrong of it, I feel a duty to let people know so that their expectations are managed. It's one thing to buy a cut-and-paste book knowing that the difference among the books is what you want.

To make a long story short, this book has some incredibly valuable information on C coding, but readers and buyers should be aware that 50% of the book could very well already exist on their bookshelf.

### DESIGNING EMBEDDED HARDWARE

*John Catsoulis*

O'Reilly, 2005. 377 pages.
ISBN: 0596007558

REVIEWED BY RIK FARROW

Embedded systems has been one of my interests for years, so when this book appeared, I wanted to read it. And although it lingered on my "to be read" stack for a while, once I got into it I found that Catsoulis writes about a difficult topic clearly, and he held my interest.

Embedded systems are everywhere, more common than desktop computers by far. I had worked with embedded systems, as well as early PC components that used co-processors, back in the early 1980s, and had assumed that things had simply gotten too complex to understand. Instead, Catsoulis explains how embedded system designs have gotten less complex, as chip designers worked to create hardware that is easier to integrate. If you think about it for a moment, it makes perfect sense that ease of use ranks right up there with capabilities, making this a natural evolution.

Catsoulis starts off gently, with chapters as basic as architecture, some assembler and Forth, and Electronics 101; there are even soldering tips. He then moves into the use of specific interconnects, the buses of embedded systems. These chapters initially caught my interest, as they explained concepts I had heard about: the Canbus used in my Prius, or I2C used with real-time clocks in PCs, and even a good explanation of USB. To me, these chapters alone were worth the price of the book. Then Catsoulis describes how these buses are used to tie together sensors, relays, and various microprocessors. Catsoulis's experience teaching college-level classes in embedded system design shows as he points out commonly made design errors, such as forgetting to use draw-down resistors.

If you have ever considered building that network-connected toaster or Web-based wine-cellar temperature sensor, this is the book for you. Even if you won't be designing your own circuit boards, you will certainly understand what is involved in any kit or prebuilt design you may decide to use.

### STEALING THE NETWORK: HOW TO OWN A CONTINENT

*131ah, Russ Rogers, Jay Beale, Joe Grand, Fyodor, FC, Paul Craig, Timothy Mullen, and Tom Parker*

Syngress, 2004. 432 pages.
ISBN 1-931836-05-1

This is another book that sat around gathering dust; I started reading it in the middle, then went on to read the whole thing. Books in this series (the titles of which start with "Stealing") are fictional accounts of hacks and hacking. The chapters vary in quality, but I found I enjoyed reading most of this book, perhaps in part because I know many of the authors and could recognize their hacking styles in their chapters. *Stealing the Network* makes for good idle-time reading.

NICHOLAS M. STOUGHTON

# an update on standards

Nick is the USENIX Standards Liaison and represesnts the Association in the POSIX, ISO, C, and LSB working groups. He is the ISO organizational representative to the Austin group, a member of INCITS committees J11 and CT22, and the Specification Authority subgroup leader for the LSB.

*nick@usenix.org*

As you know if you've been following this column, the POSIX standard is undergoing a revision. This is the third official full revision since it first became a standard in 1988. In this article, we'll take a more detailed look at some of the new interfaces that are planned for inclusion in the revised standard. There are four separate sets of new interfaces, each of which is currently an official Open Group specification.

### SET 1: GENERAL INTERFACES

There are several extremely useful interfaces in the GNU C library, glibc, many of which are also found in other vendors' libraries. These interfaces can be broadly grouped into the following categories:

- Directory handling: alphasort(), dirfd(), and scandir().
- Signal handling: psignal() and psiginfo().
- Standard I/O extensions: dprintf(), fmemopen(), getdelim(), getline(), open_memstream(), and open_wmemstream().
- Temporary files: mkdtemp().
- String handling: stpcpy(), stpncpy(), strndup(), strnlen(), strsignal(), mbsnrtowcs(), wcpcpy(), wcpncpy(), wcscasecmp(), wcsdup(), wcsnlen(), and wcsnrtombs().

I don't plan to describe each and every one of these interfaces in detail, but there are some interesting points to note. First and foremost is the relationship between this project and the Technical Report the ISO C committee is preparing on "bounds checking interfaces." Although the ISO C document contains newly invented functions to supplement the standard I/O and string handling functions of the ISO C standard, it will only be a Technical Report. This is not the same as a Standard; it is a way of testing the water, providing a trial-use period to see whether industry is interested in going that way. At this point, a few companies have indicated an interest in that approach, including both Microsoft and Cisco.

However, the interfaces listed above will be going into the POSIX standard and will have the full weight of an International Standard to them. They are not invention, and they have been implemented (quite probably on the system you are using). Many of them solve the same problem, buffer overflow, that the ISO C Technical Report tries to, but in a very different way. There is a second part to the ISO C technical report planned, which will reference many of these new POSIX interfaces as better alternatives if you are designing new programs. On the other hand, if you are retrofitting large, established code bases to fix potential buffer overflows, then the ISO C inventions may be useful.

Two interfaces fmemopen() and open_memstream(), are particularly interesting, in that they provide a way of performing standard I/O to dynamically allocated memory buffers. Consider the following:

```
char *
itos (int i)
{
    FILE *f;
    size_t len;
    char *buf;

    if((f = open_memstream(&buf,
    &len)) == NULL)
            return NULL;
    fprintf(f, "%d", i);
    fclose(f);
    return buf;
}
```

Although this is a rather trivial use of the new functionality, it serves to illustrate the point. The function converts an integer to a

string, allocating space for the string as required. A more conventional program might have chosen to use a static buffer and assumed that the size of an integer was *n* bits, and therefore the maximum length that the string could ever be was *m* bytes. And the program would have overflowed its buffer when ported to a system with a larger size of integer.

### SET 2: PATHNAMES RELATIVE TO OPEN DIRECTORIES

Solaris 10 introduced a handful of file system interfaces to work on files with extended attributes. These interfaces were all named with an ...at() suffix, and they took a file descriptor of an open directory as the first argument. Relative pathnames are relative to the open directory, and not (necessarily) relative to the current working directory. For example, openat() behaves as ordinary open(), except that it takes an additional argument, the file descriptor for relative pathnames. In the Solaris case, openat() accepts an additional value, O_XATTR, for the file mode.

In the glibc case, the extended attributes part of these interfaces was dropped, but the concept of handling pathnames relative to an open directory proved a powerful mechanism for addressing a number of security and other related issues, and so the concept was extended to all system interfaces that took a pathname. One other interface in this set is fexecve(), which is similar to execve() except that it executes the file on an open file descriptor. This allows, for example, a program to open the file it is about to execute, lock it, checksum it, and only execute it if it matches the expected checksum. Without fexecve(), an application that attempted to do this would suffer a vulnerability that the file could be replaced between successfully checksumming it and executing it.

One other useful feature of this set is the ability to avoid (or at least postpone) buffer overflow with pathnames that exceed PATH_MAX bytes.

The complete list of interfaces in this set is as follows: faccessat(), fchmodat(), fchownat(), fdopendir(), fexecve(), fstatat(), futimesat(), linkat(), mkdirat(), mkfifoat(), mknodat(), openat(), readlinkat(), renameat(), symlinkat(), and unlinkat().

One other noteworthy point must be made here: futimesat() may yet change its name and functionality slightly. There is an intention in this revision of POSIX to include file timestamps with nanosecond granularity. Until now POSIX has specified only one-second granularity on files. However, almost all OS vendors now have support for a finer-grain resolution, typically at the nanosecond level. As processors get faster and faster, the ability for tools to be able portably to distinguish between a source file and a file generated from that source becomes more and more important. Since futimesat() is a new function, both in glibc (as far as I am aware, it has not been implemented anywhere else) and POSIX, this may be the best place to add support for setting file time stamps at this fine a granularity. This aspect is still under discussion in the committee.

### SET 3: ROBUST MUTEXES

Developers of multi-threaded applications are probably well aware of the problems that can arise when a process terminates while one of its threads holds a mutex lock. While it is sometimes possible for another thread to unlock the mutex and recover its state, this is at best an unreliable and unportable mechanism.

Robust mutexes are introduced in this set of new interfaces. A robust mutex is simply a mutex with a special "robust" bit set in its attributes. Whenever a thread that owns a robust mutex terminates, current or future waiters on that mutex will be notified that the owner is dead. Another thread then has the opportunity to take over and clean up the state that was protected by the mutex and to make the mutex once again consistent.

One important feature of this proposal is that it is only intended to deal with abnormal termination of the process owning the mutex (e.g., if the process was subject to a signal). It is not intended to be a way to encourage bad programming and have applications simply not bother to clean up properly at exit, and so on; therefore, if a thread is terminated by cancellation or if it calls pthread_exit(), it is expected that that thread will handle its own cleanup properly (e.g., by registering appropriate cleanup handlers).

This set of interfaces includes pthread_mutex_consistent(), pthread_mutexattr_getrobust(), and pthread_mutexattr_setrobust(). It also alters the behavior of several other existing mutex APIs, essentially by adding the EOWNERDEAD error return.

### SET 4: THREAD-AWARE LOCALES

The concept of locales to allow processes to have different natural-language interfaces has always been a part of POSIX. Until now, the process has been the object that is associated with a locale. This set of new APIs permits individual threads to be in different locales.

The major new concept in this set of interfaces is the locale_t object. Applications can create as many locale objects as they require, each one associated with a different locale. Each thread can then choose to use one of these locales, and in doing so does not affect the behavior of any other thread. Compare this with the old concept of the process as a whole being in a given locale; if one thread changed the locale, then all the threads in that process would be changed.

The fundamental interfaces in this set are newlocale(), duplocale(), freelocale(), and uselocale().

In addition to these, all of the ctype.h character categorization functions gain a new locale object counterpart. For example, as well as isalnum(int c), there is an isalnum_l(int c, locale_t l) interface. The former returns true if the character represented by c is alphanumeric. The new interface returns true if the character is alphanumeric in the locale represented by l.

### THE TIMETABLE

The revision project has been working up to full steam over the past couple of years, but it is now in full-scale development mode. The first committee drafts appeared in July (as I write this article). The second draft, which will probably be the first one to be publicly balloted, is scheduled for November 2006. The document will probably take until April 2008 before it is completely approved. As always, the Austin Group welcomes any interested party to join the process. For details, see www.opengroup.org/austin.

# 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, Java, 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/.

**ACCESS TO PAPERS** from USENIX conferences online: www.usenix.org/publications/library/proceedings/

**THE RIGHT TO VOTE** on matters affecting the Association, its bylaws, and election of its directors and officers.

**DISCOUNTS** on registration fees for all USENIX conferences.

**DISCOUNTS** on the purchase of proceedings and CD-ROMs from USENIX conferences.

**SPECIAL DISCOUNTS** on a variety of products, books, software, and periodicals. For details, see www.usenix.org/membership/specialdisc.html.

**TO JOIN SAGE,** see www.usenix.org/membership/classes.html#sage.

**FOR MORE INFORMATION** regarding membership or benefits, please see www.usenix.org/membership/ or contact office@usenix.org. Phone: 510-528-8649

## USENIX BOARD OF DIRECTORS

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

PRESIDENT
Michael B. Jones,
*mike@usenix.org*

VICE PRESIDENT
Clem Cole,
*clem@usenix.org*

SECRETARY
Alva Couch,
*alva@usenix.org*

TREASURER
Theodore Ts'o,
*ted@usenix.org*

DIRECTORS
Matt Blaze,
*matt@usenix.org*

Rémy Evard,
*remy@usenix.org*

Niels Provos,
*niels@usenix.org*

Margo Seltzer,
*margo@usenix.org*

EXECUTIVE DIRECTOR
Ellie Young,
*ellie@usenix.org*

## LETTERS TO THE EDITOR

### TO ROBERT HASKINS

Robert,

I just read "ISPadmin: Anti-Spam Roundup" in the October issue of *;login;*. I thought you might have been a bit more clear about reputation filtering with regard to DCC. DCC has offered reputation filtering with its commercial license for close to a year now. Vernon tests Solaris builds on USENIX's MX, so I don't know whether the commercial license is revenue-generating. You might ping Vernon about it.

Otherwise I enjoyed your column, as always.

*Tony Del Porto*
*Sysadmin, USENIX Association*

### ROBERT HASKINS REPLIES

Tony is indeed correct. The Rhyolite commercial solution Tony mentions combines IP reputation information from originating IP addresses with the DCC checksum data from messages, using data from paying subscribers only. Utilizing paying clients helps to eliminate the possibility of the checksum data getting contaminated by spammers wanting to get their junk through.

### TO MARK BURGESS

Dear Sir,

To me *;login:* seems to reinvent the wheel in about every issue.

One of the other articles in *;login:* February 2006 is about Configuration Management. That is part of ISO 20000, so why reinvent it? In fact several other articles in earlier *;login:*s refer to problems and issues that are addressed in a structured manner (engineering-like) in ISO 20000, ISO 17799, and other international standards. I think it will benefit your magazine, and ICT in general, to refer to international standards when they are available. The standards are not absolute, so any discussion will benefit the community.

A practical suggestion: perhaps a thorough article on ISO 20000 (ITIL) may be a suitable start. There is a U.S. chapter of the user group, www.itsmf.com.

ISO 20000 was originally developed by users; that is one reason why it is useful. However, there are a number of tools that follow

the standard and may help implementation.

Until ICT quality is much improved, we are seen as hackers, not professionals.

It would be nice to have an answer, perhaps that I and ISO are both junk?

*Yours very truly,*
*Tore Audun Høie,*
*Ph.D. computer science*

#### MARK BURGESS REPLIES

My answer is simple:

ISO17799 (formerly BS17799) is a standard for heuristic security management in organizations, and ISO20000 (formerly BS15000) is the ITIL reference document. Both of these are high-level, handwaving guidelines about service and business operations. True enough, they pay lip service to configuration management, but the configuration management they refer to is not the same as discussed in *;login:*—it concerns software and information organization and revision/change control. The technical problem covered in *;login:* is more about automation, tuning, and maintenance in operating systems. The principles are somewhat similar, but these ISO documents offer no solutions to implementation, only finger-wagging "should do's" to be complied with.

## ADDENDUM TO ANNUAL TECH '06

#### 2006 USENIX ANNUAL TECHNICAL CONFERENCE INVITED TALK

### *Hackers and Founders*

*Paul Graham, Y Combinator*

*Summarized by Marc Chiarini*

*[This summary was inadvertently omitted from the conference summaries published in the October 2006 ;login:. We apologize to the summarizer, the speaker, and our readers.]*

Graham, a well-known hacker and essayist (and all-around nice guy) gave a thought-provoking and at times hilarious talk about the power of the marginal. He began with an observation by his friend Trevor Blackwell. On a trip to the Apple garage, Blackwell who hails from Saskatchewan, was amazed at how dedicated Jobs and Wozniak must have been to work in a garage. "Those guys must have been freezing!" Graham pointed out that the mild climate of Silicon Valley, which has sprouted quite a few famous startups, encourages work on the margins, where there is more incentive to tinker and much less need to justify the use of well-heated indoor spaces. There is a paradox, however: even though many hackers and founders come from and work best on the margins, many also crave acceptance by the mainstream. This is not a good thing; most great ideas come from the margins. Graham made a witty attempt at explaining why this is so and what can be done to encourage the process.

He touched on many core ideas: the disadvantages of "insider" (mainstream) projects, illustrated via analogy with the government commissioning the writing of the Great American Novel; ways of determining in what fields it's worth trying to become an insider, including evaluation of the tests that admit you and the quality of existing insiders (from a practitioner's point of view); why big companies frequently get blind-sided by startups, because the employees continually undergo tests for the wrong qualities; how outsider success hinges on corrupt tests selecting ineffectual insiders with lots of money, followed by fair tests such as the marketplace, where, thanks to the Internet, ideas are increasingly promotable on a level playing field.

Graham provided a veritable guidebook for success as an outsider: In any field, even in those with honest tests for inner-circle admission, outsiders don't have much to lose; they can take risks again and again, with few people noticing their failures. Tradition should generally be shunned, as the state of the art changes much faster these days and the space of possibilities is ever growing. Nor can outsiders allow their lives to become scheduled; it's not good for thinking. Long, uninterrupted blocks of time allow broad tinkering. It's also essential for outsiders to stay in direct contact with the latest platforms, programming languages, and other technologies. Delegation, especially in the starting phases of an "unplanned" project, is a death knell; if you are not doing almost all the work yourself, you stop learning. Outsiders must find problems that can be solved in one person's head (like the Woz building the hardware and software for the Apple II). One way is to focus on the places where tasks are normally divided: create a programming language and, instead of shotgunning it to other hackers, build something useful with it and hand that off. Since outsiders don't have the benefit of highly focused training, they can cast a wide net, creating new interdisciplinary projects for themselves, learning enough in each area to hack together something brand new. Finally, working on small things provides quick gratification and the ability to make do with less.

The remainder of Graham's talk focused on how to make up for what insiders often have—for instance, an audience, money, nonmaterial resources—without becoming like them. His concluding advice was to try just hacking things together; when people complain that you're

unqualified or that what you've been doing is "inappropriate," you know you're on the right track!

In the Q&A, people asked what it takes to be a good startup founder. You need to be unbelievably determined, you have to have a good sense of design, and you have to be outgoing enough to speak with other people. Q: How does one make something marginal catch on? A: Start with other hackers and early adopters (Google was a great example, no marketing, just word of mouth). Q: What is the path to startup success? A: The most important thing is to make something that other people want or, better yet, need. Q: How do you know when to let something you've created run its course or to intervene in its development? A: You cannot hose yourself by open-sourcing everything and letting people play. Q: How do you know when something has failed and it's time to try your next foolish idea? A: Collect good friends whose opinion you trust, and always be open to suggestions.

## THANKS TO OUR VOLUNTEERS

*Ellie Young*
*ellie@usenix.org*

As many of our members know, USENIX's success is attributable to a large number of volunteers, who lend their expertise and support for our conferences, publications, and member services. They work closely with our small staff in bringing you the best there is in the fields of systems research and system administration. Many of you have participated on program committees, steering committees, and subcommittees and in SAGE, as well as contributing to this magazine. We are most grateful to you all. I would like to make special mention of the following individuals who made significant contributions in 2006.

*The program chairs for our 2006 conferences:*

Larry Peterson and Timothy Roscoe, NSDI '06

Atul Adya and Erich Nahum, 2006 USENIX Annual Technical Conference

Mahadev Satyanarayanan and Nigel Davies, MobiSys 2006

Steven M. Bellovin, SRUTI '06

Matt Blaze and Angelos D. Keromytis, First HotSec Workshop

Dan Wallach and Ron Rivest, first Electronic Voting Technology Workshop

Angelos D. Keromytis, USENIX Security '06

Ted Ts'o, 2006 Linux Kernel Developers Summit

David Andersen and Neil Spring, WORLDS '06

Brian Bershad and Jeff Mogul, OSDI '06

Michi Henning and Maarten van Steen, Middleware 2006

William LeFebvre, LISA '06

*Invited Talks/special track chairs:*

Chris Small and Matt Blaze, Invited Talks for 2006 USENIX Annual Technical Conference

Patrick McDaniel and Gary McGraw, Invited Talks for USENIX Security '06

David N. Blank-Edelman and Doug Hughes, Invited Talks for LISA '06

Philip Kizer, Guru Is In Coordinator for LISA '06

*Some other major contributors:*

Balachander Krishnamurthy for his continued efforts in obtaining sponsorships and providing guidance for SRUTI

Alva Couch for liaising with VEE and HotAC, co-sponsored by USENIX

Avi Rubin and ACCURATE for helping organize our first Electronic Voting Technology Workshop

Peter Honeyman for his efforts in outreach to the international community, e.g., the SANE and Middleware conferences

Michael B. Jones for serving as liaison to the Computing Research Association

Matt Blaze, Clem Cole, Alva Couch, Rémy Evard, Jon "maddog" Hall, Michael B. Jones, Marshall Kirk McKusick, Niels Provos, Margo Seltzer, and Theodore Ts'o for their service on the USENIX Board of Directors in 2006

Jon "maddog" Hall for holding auctions for contributions to the John Lions Chair in Operating Systems at the University of New South Wales

Dan Geer, Theodore Ts'o, and Marshall Kirk McKusick for serving on the USENIX audit committee

Clem Cole, Peter Salus, Keith Packard, John Gilmore, Jim McGinness, and Jon "maddog" Hall for serving on the USENIX awards committee

Rob Kolstad and Don Piele for their work with the USA Computing Olympiad, co-sponsored by USENIX

## SAGE UPDATE

Greetings from USENIX. We've been busy, working hard on the upcoming LISA conference and a fantastic new SAGE Web site. Take a look at what's been happening with SAGE.

### LISA ONSITE REGISTRATION, GROUP DISCOUNTS, AND COMMUNITY MEETINGS

The LISA Program Committee and USENIX have put together a high-caliber slate of tutorials for LISA '06. There's something for everyone, whether their specialty is storage, networks, security, or jack-of-all-trades sysadmin. It's not too late to register—in fact, it's the perfect time, lest you find that your annual conference fell victim to somebody's Q4 budget squeeze. Onsite registration opens at 5 p.m. on Saturday, December 2. Bring your manager and your colleagues, and qualify for the multiple-employee discount by taking 5 or more people to LISA: http://www.usenix.org/events/lisa06/.

### JOIN US AT LISA—SAGE COMMUNITY MEETING ON DECEMBER 6

We've scheduled the SAGE Community Meeting for Wednesday night, December 6, and we hope to see a large presence there. We value the input you have given us to keep SAGE on track, so please keep it coming.

### NEW SAGE WEB GOES LIVE

It's here! The new SAGE Web site is live, and it's a lean, clean, information machine! We listened to your feedback and gave you the same no-nonsense interface that works so well on the USENIX site, with all the features you expect from SAGE: Jobs Board, Speakers Bureau, and so on. The new site is more than just skin-deep, though: Content has been updated and expanded throughout, and new functionality has been written in. Check it out and learn more about SAGE at http://www.sage.org.

### JOHN LIONS FUND—LAST CALL

You may recall that in the April issue (p. 80) we announced that USENIX was matching donations to the fund to establish an endowed Chair in Operating Systems at the University of New South Wales. The period of matching donations is rapidly drawing to a close.

To double the value of your contribution, make your donation before December 31, 2006. Send a check to:

John Lions Fund
USENIX Association
2560 Ninth St., Suite 215
Berkeley, CA 94710

or donate online at
http://www.usenix.org/about/lionsfund/

**Statement of Ownership, Management, and Circulation, 10/2/06**

Title: *;login:* Pub. No. 0008-334. Frequency: Bimonthly. Subscription price $115.
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 published nearest to filing date of 10/2/06* |
|---|---|---|
| A. Total number of copies | 6800 | 7140 |
| B. Paid and/or requested circulation | | |
| Outside-county mail subscriptions | 3977 | 4105 |
| In-county subscriptions | 0 | 0 |
| Other non-USPS parcel distribution | 1726 | 1793 |
| Other classes | 0 | 0 |
| C. Total paid and/or requested circulation | 5703 | 5898 |
| D. Free distribution by mail | | |
| Outside-county | 0 | 0 |
| In-county | 0 | 0 |
| Other classes mailed through the USPS | 61 | 73 |
| E. Free distribution outside the mail | 555 | 720 |
| F. Total free distribution | 616 | 793 |
| G. Total distribution | 6319 | 6691 |
| H. Copies not distributed | 481 | 449 |
| I. Total | 6800 | 7140 |
| Percent Paid and/or Requested Circulation | 91% | 89% |

I certify that the statements made by me above are correct and complete.
Ellie Young, Publisher

# conference reports

THANKS TO OUR SUMMARIZERS

*SECURITY '06*

Madhukar Anand
Tanya Bragin
Kevin Butler
Lionel Litty
Michael Locasto
Bryan D. Payne
Manigandan Radhakrishnan
Micah Sherr
Patrick Traynor
Wei Xu

*METRICON 1.0*

Dan Geer

*NSPW '06*

Matt Bishop
Michael Collins
Carrie Gates
Abe Singer

CONTENTS



## Security '06: 15th USENIX Security Symposium

*Vancouver, B.C., Canada*
*July 31–August 4, 2006*

KEYNOTE ADDRESS

***The Current State of the War on Terrorism and What It Means for Homeland Security and Technology***

*Richard A. Clarke, Chairman, Good Harbor Consulting LLC*

*Summarized by Kevin Butler*

Richard Clarke gave an impassioned, scathing indictment of the U.S. government's policies and practices that electrified the crowd in Vancouver, bringing them to their feet at the conclusion of his keynote speech. Clarke was the former counterterrorism advisor on the U.S. National Security Council for the Bush administration and has advised every administration since Reagan. He brought his 30 years of consulting work to bear on his speech, where he laid out many of the issues facing both the security research community and the populace at large, given the current global situation.

Clarke began by commenting that, nearing the fifth anniversary of the September 11 attacks, we should examine what measures to increase security have been addressed since that time. As Clarke pointed out, five years is a long time: It took less than five years for the Allies to destroy Nazi Germany and Imperial Japan. The day after the 9/11 attacks, President Bush asked Clarke to outline a series of plans to maintain security. Clarke responded with a series of blueprints for going after Al Qaeda while simultaneously addressing vulnerabilities at home. Part of these plans concerned the state of IT security, given the growing dependence of the nation on cyber-based systems. Unfortunately, since that time, little of what had been discussed was implemented, while other, more draconian measures that impinge on civil liberties have instead taken their place.

Much of Clarke's criticism was aimed at the administration's handling of the Al Qaeda threat. After 9/11 there was talk about finding the cells and taking out the leadership, when the focus should have also included attacking the root causes of problems: a battle of ideas, where the West displays a better ideology that is more appealing to the Islamic world than that promulgated by Al Qaeda. The results have not been great to this point, as Al Qaeda still exists and has transformed from a hierarchical, pyramid structure to a decentralized organization with dozens of individual organizations. In the 36 months after 9/11, Clarke asserted, the number of attacks by groups related to Al Qaeda doubled around the world. Clarke satirically referred to an Arabic satellite TV station established by the U.S. government "that nobody watches" as our entry into the battle of ideas.

He also showed many similarities between the American occupation of Iraq and the French occupation of Algeria, a conflict that the French ultimately lost. By alienating the Iraqi population thanks to the stories from Abu Grahaib, the assaults on Fallujah, and the perception of killing innocent civilians in Iraq (despite what the truth may be, Al Jazeera and Al Arabia show daily images of Americans killing women and children through aerial and other assaults), we are losing the battle of ideas and convincing the Iraqis that Al Qaeda was right: For its oil, we are taking over a country that did nothing to us. We are almost fulfilling bin Laden's prophecies. What are the metrics of success in Iraq? Clarke sug-

gested these are the economy and stability of the country. When asked how many innocent civilians had been killed in Iraq, President Bush replied, "Around 30,000"; however, a team from Johns Hopkins did a field survey using internationally accepted metrics and found the number was close to 100,000, not counting the 2,500 dead and 10,000 wounded Americans. The Pentagon admits to having already spent $400 billion on the war, and some estimates put the total price tag at over a trillion dollars. Part of the rationale of the high economic and human cost is that fighting the terrorists "over there" means we won't be fighting them "here." This argument is logically fallacious, however; nothing we are doing there prevents them from fighting us here or anywhere else, as evidenced by the attacks on the Madrid train system, attacks on the London subway, and the planned attacks in Toronto. None of the fighting overseas prevented these situations, nor will it prevent future attacks in Canada and the United States.

Clarke argued that more effort should have been spent protecting critical infrastructure and minimizing the possibilities of damage here. An ABC report showed how easy it was to take a backpack onto a train and leave it unattended, the modus operandi of the Madrid attacks. Similarly, 120 chemical plants in the United States store lethal gas, with over a million people in the plume radius. If one such plant were attacked and a plume unleashed, over 17,000 casualties would be expected; however, Congress has debated plant protection for over three years but nothing has been done. This has occurred with issue after issue. Instead, measures that reduce civil liberties—such as the ineffective color-coded threat system—have been implemented by abusing the term "security." The government engaged in mass wiretapping without any judicial oversight and, combined with other abuses, this has led to the association of security with Big Brother in the minds of the public.

The current situation has implications for security researchers, as currently many systems were not designed with security in mind, and Clarke asserted that research into securing these systems is necessary. However, DARPA no longer funds many of these activities, and research money is being handled by the Department of Homeland Security through the HSARPRA program; however, this program is severely underfunded, with this year's budget cut from $16 million to $12 million—Clarke noted that such was the administration's commitment to cybersecurity research. This view is shared by the president's own committee, which identified the program as requiring significant new funding, but this has not yet come to pass.

Clarke made some controversial statements describing what regulation should be required. He suggested that identifying best practices for ISPs is possible to regulate the service provider industry. He also suggested governmental regulation in critical systems such as electrical power systems and banking, but he considers the current government ideologically opposed to such measures. The upshot is that many opportunities to focus on real security have been wasted. The best thing for us in the community to do is not to rely on the government, but to unite and come up with ways of solving problems ourselves, as well as staying vigilant for governmental attempts to further erode civil liberties. He ended his address by noting that when people erode civil liberties, privacy, and constitutional guarantees, they need to be reminded of their oath to defend the Constitution against all enemies, lest they become the enemies themselves.

A spirited question period followed. To the question "Is it that the administration is apathetic, malicious, or doesn't have a clue?" Clarke responded, "Yes." The administration, in his view, cannot be educated and needs to be replaced. He also reiterated the need to win the battle of ideas by supporting moderate and progressive voices in the Middle East, and he noted the amount of "security theatre" being practiced, where there is a show of security without substance. A particularly resonant point was the need to find and disclose vulnerabilities; as Clarke noted, our real enemies already know our vulnerabilities and security processes, and pretending we won't talk about them because these enemies will learn about them is very wrong. One of the best defenses against future attacks is an environment of openness and disclosure. Another important point was that we need to accept that risk is present and casualties will occur; the question of an attack is not "if" but, rather, "when." We should focus on mitigating the effects of the next attacks whenever they may happen, while focusing on ideals such as due process and guaranteed Constitutional rights that are the hallmarks of our civilization: If we cannot preserve those, then what are we fighting for?

## AUTHENTICATION

*Summarized by Micah Sherr*

### *A Usability Study and Critique of Two Password Managers*

*Sonia Chiasson, P.C. van Oorschot, and Robert Biddle, Carleton University*

Sonia Chiasson presented a usability study of two password managers, PwdHash (USENIX Security '05) and Password Multiplier (WWW2005). Although both password managers attempt to increase security by shifting the burden of creating and maintaining strong passwords away from the user, the study demonstrated that usability issues in the applications led to incorrect usage of the managers and, in some cases, to the leakage of password information.

Sonia presented the results of a follow-up questionnaire given to members of the focus group. Participants indicated that they did not perceive an improved sense of security. Users were uncomfortable with not knowing the passwords to the sites they visited, and many had misconceptions as to the operation of the managers (e.g., the belief that their passwords were kept in a large and remote database). At the same time, the transparency of the applications often led to false senses of security. For example, users who installed a password manager but failed to activate it falsely believed their passwords were being protected. Sonia concluded by noting that the usability of a system directly impacts its security. Security systems must support users' conceptions as to how software should operate; they should imbue a sound (and not necessarily transparent) sense of security.

### *On the Release of CRLs in Public Key Infrastructure*

*Chengyu Ma, Beijing University; Nan Hu and Yingjiu Li, Singapore Management University*

The talk was given by Yingjiu Li, who presented a method of determining a more nearly optimal schedule for distributing certificate revocation lists (CRLs)—time-stamped lists of certificates that have expired or become compromised or otherwise invalidated. In the first part of his talk, Yingjiu described an empirical study of CRL distributions. By examining CRL release data from VeriSign, he and his coauthors derived a probability distribution function for the distribution of CRLs. They determined that most revocation requests occur within the first few days after a certificate is issued and are less common as the certificate ages.

In the second part of his talk, Yingjiu introduced a new economic model for optimizing the distribution of CRLs. The objective of the model is to minimize the total operational cost of distributing the certificates while maintaining a reasonable degree of security. Yingjiu presented evidence that different types of certificate authorities (CAs)—i.e., start-up CAs and well-established CAs—should adopt different strategies for CRL distribution and that the number of CRL distributions will stabilize after a period of time.

### *Biometric Authentication Revisited: Understanding the Impact of Wolves in Sheep's Clothing*

*Lucas Ballard and Fabian Monrose, Johns Hopkins University; Daniel Lopresti, Lehigh University*

Lucas Ballard presented a study of biometric authentication, a method of using biological and physiological traits to authenticate humans. In particular, Lucas's talk focused on exposing weaknesses in both the implementations and evaluation methods for biometric authentication schemes.

The presentation examined the security of biometrics based on the writing of passphrases (i.e., a human is authenticated by comparing his or her passphrase against a corpus of previously supplied handwriting samples). Lucas presented results that show that even novice forgers (students who showed some talent in producing forgeries) can produce forgeries far better than what the biometric systems predicted could be produced.

In the last part of the talk, Lucas described a system for automating forgeries based on a generative model. A synthesis algorithm selects n-grams from a corpus of the target's handwriting samples. The n-grams are then used to forge a passphrase in the target's handwriting style. Lucas presented results that show that the generated and forged passphrases perform better than those generated by skilled forgers.

## INVITED TALK

### *Selling Security to Software Developers: Lessons Learned While Building a Commercial Static Analysis Tool*

*Brian Chess, Fortify Software*

*Summarized by Tanya Bragin*

Brian Chess said that finding security vulnerabilities in source code is like trying to find the proverbial "needle in a haystack." However, he stressed that well-built, highly configurable tools not only help organizations uncover potential problems but also shape long-term software development policy and its enforcement.

There are several reasons static analysis works well. First, existing software development proce-

dures can easily be modified to employ source code analysis and in doing so allow bugs to be fixed before deployment. Second, there are a limited number of common vulnerability types, so patterns can be used to detect them via static analysis methods. Third, static analysis tools can be built to give uniform coverage over large code bases in reasonable time, as opposed to, for example, those using dynamic analysis, and thus they are practical.

Although there have been many academic efforts to develop static analysis techniques, they are not appropriate for use in commercial environments, for a number of reasons. First, the tools developed in academic settings were not explicitly designed for the scale and variety of environments in which they could be used. Consequently, they generally lack the customizability needed in a complex organization. Second, reporting in such tools tends to be insufficient for enterprise use. Finally, the management interfaces required for over-time tracking of defects and activities is lacking.

It turns out that these shortcomings are key to the eventual success of a practical static analysis tool. Aside from simply working well, finding important vulnerabilities, not crashing, not hanging, and not having any vulnerabilities of its own, the tool has to be scalable, well-documented, and intuitive to a person who is not an expert in all nuances of software security. The main users of static analysis tools in industry tend to not just be developers themselves, but security auditing departments that outline, monitor, and enforce organization-wide security policies, and Brian stressed the importance of working closely with such teams.

Adopting new technology that fundamentally changes how software should be written is hard and can cause resistance from developers. "Developers are optimizers," stated Brian, himself having come from that background. "If they think that security is optional, they will optimize it out." He continued with a couple of amusing examples of excuses he has heard out in the field about why obvious vulnerabilities do not actually make code insecure, such as "I trust the system administrator" or "That code will never be run." But in reality these vulnerabilities can open unexpected backdoors into critical software, which significantly damage the company brand, open it up to liability, and cause economic losses.

Teaching developers to think about security is critical, concluded Brian, but we need to enable them to do their job efficiently with the necessary tools. A good analogy is spelling, he offered. We teach everybody to spell, but we still have spell checkers in our word processors. Similarly to how we already have compilers to check for proper programming language syntax, static analysis tools can go a step further and help enforce good programming practices that result in fewer software vulnerabilities and incrementally help developers write good code on their own.

## INVITED TALK

### *Security Vulnerabilities, Exploits, and Attack Patterns: 15 Years of Art, Pseudo-Science, Fun, and Profit*

*Ivan Arce, Core Security Technologies*

*Summarized by Madhukar Anand*

In conferences such as the USENIX Security Symposium, it is typical to learn about the work and experience of academicians and researchers in computer science. The invited talk by Ivan Arce (CTO of Core Security Technologies) was very unlike this. It offered an insightful, alternate perspective on the evolution of information security. Arce presented his views based on his forays and experiments in designing systems for fun and profit.

Arce, introduced to the world of computers through the Commodore VIC 20, recounted how he saw computers as a toy to experiment with. Consequently, he grew up with a notion of computers as a game rather than a tool. Programming the VICs taught him that computers could be tailored and have many hidden features, and he learned the key difference between an enemy and an adversary. Some of these experiences have shaped his views on the evolution of attacks and security paradigms in the past 15 years.

Evolution of Attacks: In the early 1990s (post–RT Morris worm) there was no Linux, TCP/IP stack in Windows, or the Web. Security information flowed from technical journals, bulletin boards, and underground publications. For most people outside of the academic world, these underground publications were the main source. This was the time when many home-computer users started turning professional—kids who picked up programming, started exploring their home PCs, and turned to security for jobs. By the mid-1990s, exploits in shell code become common and stack smashing was done for fun and profit. From then on, there has been increasing complexity in software and, subsequently, an increasing sophistication in attacker skills.

Today, the attacker has the ability to hack networks, write viruses, and reverse-engineer software. The trend now, post-2001, is to go directly at a workstation and

own the whole network. This works because there are myriad vulnerable applications, it is difficult to implement inventory, it is difficult to deploy and manage countermeasures, desktops are operated by careless and unaware users, and, above all, it represents the law of "minimal effort for maximum profit."

Arce observed that in our "quest for completeness," we have too strongly emphasized understanding attacks, rather than vulnerabilities. To illustrate this point, he gave the example of the ssh v1 CRC insertion attack discovered in 1998. The patch distributed to fix this vulnerability was itself found to contain a bug. Therefore, it is important that we focus on getting the basics right rather than going after the obscure and complicated.

Another problem with current systems is that underlying models of systems have not kept up with changing technology. The management, deployment (policies, ACLs, RBAC, authentication tokens, etc.), and generation of security value (AV/IDS signatures, patches, certificates, vulnerability checks, etc.) in an information system are centralized. In contrast, outside of the information world, there has been an evolution of open source software, P2P, mobile code, and technology based on social networking and reputation- and collaboration-based systems. So the current information security technology and business models should not ignore them.

In conclusion, Arce felt that the generation that entered the information security field in the early 1990s has successfully managed to create a market for security. This generation has embraced and promoted open and unmediated discussion about security. He felt that we are better off now than we were 15 years ago (at least, we know where we are wrong), and so he urged everyone to learn from past mistakes, stop reinventing the wheel, and brace ourselves for a new generation in information security.

## ATTACKS

*Summarized by Patrick Traynor*

### ■ *How to Build a Low-Cost, Extended-Range RFID Skimmer*

*Ilan Kirschenbaum and Avishai Wool, Tel Aviv University*

Ilan Kirschenbaum explained that as radio-frequency identifier (RFID) technology begins to permeate our lives (credit cards, passports, etc.), many in the community have voiced their concerns about the security of such devices. An attacker can, with little difficulty, skim the data entrusted to these devices from a distance of tens of meters with little effort. Unlike with other networking technologies, however, no one has examined the challenges of building a device capable of such a feat. To address these issues, Ilan discussed the process of developing a portable, extended-range ISO 14443-A compliant RFID skimmer. Using a Texas Instruments RFID reader and one of two antennas (a 10x15 cm printed PCB antenna and a 39 cm copper-tube antenna), the researchers were able to develop a leaching device capable of capturing RFID data from a distance of 35 cm, or approximately 3.5 times the advertised operational range. At approximately US$100, such a mechanism is easily within the budget of any dedicated adversary.

Many of the questions addressed the use of a loop antenna, which inherently has poor range. Additionally, because of the size of the copper-tubing antenna, many in the audience wondered about the practicality of using such a device without it being noticed. In its current form, Ilan argued, the device would in fact be extremely effective if the attacker could hide it well before the time of attack (e.g., behind drywall or around a potted plant). Ilan then concluded by mentioning that although the current work was simply a proof of concept, better antenna technology could easily be applied for a modest increase in cost.

### ■ *Keyboards and Covert Channels*

*Gaurav Shah, Andres Molina, and Matt Blaze, University of Pennsylvania*

***Awarded Best Student Paper***

Gaurav Shah pointed out that there are uncountable ways to extract sensitive data from a targeted machine. Most of these techniques, from "shoulder-surfing" to the installation of spyware, are easily detectable given current techniques. Shah discussed the JitterBug, a PS/2 keyboard attachment designed to capture such information. Unlike previous hardware schemes, which suffer from problems in recovery, the JitterBug delivers compromised data through an interpacket covert channel. Based on the time between two packets, an adversary located somewhere between the legitimate sender and the receiver can decode single bits of information without arousing suspicion on either end of the communication. Guarav stressed that such attacks are not limited to password stealing. By placing such hardware into machines prior to their distribution, adversaries from rival corporations to watchful governments can easily establish surveillance over a number of targets.

When asked how the JitterBug determined which data to transmit, Guarav discussed the use of trigger sequences. For example, attacks targeting user passwords can be activated after "ssh hostname.domain" is typed into the

keyboard. Others asked about the use of more robust TCP covert channel encoding methods that have been implemented in software. Guarav agreed that such encoding mechanisms could be incorporated into the JitterBug and that the use of interpacket spacing was sufficient for a proof of concept. Guarav concluded with a discussion on the need for error correction and repeated messages in order to account for network jitter and uncertainty. By publishing this work, the authors hoped to draw attention to the need for all devices to be part of the trusted computing base in order for real security to be attained.

■ ***Lessons from the Sony CD DRM Episode***

*J. Alex Halderman and Edward W. Felten, Princeton University*

When Sony, the world's second largest music company, deployed digital rights management (DRM) protection for its music, it also released a number of critical vulnerabilities into the digital landscape. J. Alex Halderman discussed how he and Edward Felten of Princeton University logged the events as the first major rollout of mandatory DRM software occurred. The software, made by First4Internet and SunnComm, works by automatically installing itself on the first insertion of a CD-ROM. Both products installed themselves as rootkits to each client's machine, in order to prevent their subsequent uninstallation. Unfortunately, because of vulnerabilities in both products, other malicious programs could then exploit the DRM software to gain root control. Sony initially refused to offer uninstallation software, even when presented with the weaknesses discovered by numerous researchers. However, after numerous class action lawsuits and much public duress, the music distributor eventually provided the tools necessary to rid infected machines of these rootkits.

Questioning by the audience was limited, focusing largely on the response from other members of industry. For example, a number of attendees were interested in whether or not the software by First4Internet and SunnComm has been classified as spyware by antivirus companies. The speaker said that he believed that all major antivirus manufacturers in fact now classify this software as such. Halderman then discussed how the elimination of enabling mechanisms such as AutoRun would go a long way to prevent a repeat of such an incident. In the end, this work highlights the conflict between entities attempting to protect their intellectual property and legitimate users of that content. In an attempt to prevent "unapproved" use of their music, Sony in fact broke into and endangered its clients' PCs. Because the use of DRM is effectively an attempt to undermine a user's control of his or her own computing apparatus, its use is fundamentally a security issue.

## SOFTWARE

*Summarized by Lionel Litty*

■ ***Milk or Wine: Does Software Security Improve with Age?***

*Andy Ozment and Stuart E. Schechter, MIT Lincoln Laboratory*

Andy Ozment presented the results of a study that examined whether the reporting rate of security vulnerabilities decreases over time. An earlier study by Eric Rescorla had found no evidence that this was the case for the four operating systems he studied. The authors of the study found otherwise by scrutinizing OpenBSD over a period of 7.5 years, starting with version 2.3, the "foundational" version for this study. They carefully examined the 140 security advisories released over that period of time, as well as the OpenBSD source code repository, to determine exactly when a vulnerability was introduced in the source code and when it was fixed. In addition, they estimated the amount of new code introduced by each release of OpenBSD to find out whether there was a correlation with the number of vulnerabilities that were introduced in a release.

Andy reported that a majority of the vulnerabilities found during the past 7.5 years were already present in the foundational version and that the median lifetime of those vulnerabilities was at least 2.6 years. In addition, he showed that the rate at which vulnerabilities were discovered in the foundational version decreased over time, with only half as many vulnerabilities reported during the second half of the time period. Using a reliability growth model, the authors also estimated that 42 vulnerabilities remained in the foundational version. Andy concluded by saying that they found no clear correlation between number of lines of code added between versions and number of new vulnerabilities, and that the OpenBSD source code was indeed like wine.

When asked whether he thought the findings would extend to other operating systems that may have less of an emphasis on security, Andy replied that his stab in the dark was that it may, but that the decrease rate was probably slower. Theo de Raadt asked (via IRC and Dug Song) whether the study took into account the mitigation mechanisms, such as having a nonexecutable stack, added by Open-

BSD since the foundational version. Andy said that they had not and that taking them into account would make the picture look even better for OpenBSD.

### ■ *N-Variant Systems: A Secretless Framework for Security Through Diversity*

*Benjamin Cox, David Evans, Adrian Filipi, Jonathan Rowanhill, Wei Hu, Jack Davidson, John Knight, Anh Nguyen-Tuong, and Jason Hiser, University of Virginia*

Artificial diversity consists of introducing differences in the way software is executed on a machine, for example by randomizing the memory layout. This may foil an attacker who is trying to take control of the machine, as long as the attacker does not know the key used to randomize the memory layout. Benjamin Cox described a new technique that would eliminate this reliance on a secret by running several variants of the same process in parallel, resulting in provable security. For all benign inputs, the variants would behave identically, but for some attacks, the variants would diverge. A monitor would detect the divergence and raise an alarm.

Benjamin discussed two ways to introduce diversity between the variants: partitioning the address space and tagging instructions. When partitioning the address space, the two variants use distinct memory addresses. If an attack relies on the absolute memory address of either code or data, it will cause a memory fault in at least one of the variants. Instruction set tagging adds a tag to each instruction. This tag is checked and removed by a dynamic binary translator before the code is run. Any code injected by the attacker will have an incorrect tag for at least one of the variants, once again raising an alarm. Depending on the workload and the diversification technique used, overhead for the prototype ranges from 17% to more than double the execution time for Apache.

Ron Jackson questioned whether security was provable for the address space partitioning scheme. He described an attack consisting of only overwriting some of the bits of an address, allowing the attack to be successful in both variants. Benjamin answered that this attack simply fell outside the class of attacks prevented by address space partitioning. Other limitations of the current prototype, left for future work, included handling signals and shared memory.

### ■ *Taint-Enhanced Policy Enforcement: A Practical Approach to Defeat a Wide Range of Attacks*

*Wei Xu, Sandeep Bhatkar, and R. Sekar, Stony Brook University*

Wei Xu observed that before performing a security-sensitive operation, an application should check that this operation is not under the control of an attacker. This can be done by tracking what parts of an operation are tainted, i.e., originating from untrusted interfaces. Data in memory originating from the network is marked as tainted and taint is then propagated throughout the execution of the application. To this end, the source code of the application needs to be automatically instrumented to maintain a bitmap of tainted memory locations.

When a security-sensitive operation is performed, a policy-defined check is conducted to make sure the operation is safe. For instance, an application may construct a SQL query based on user input. To prevent SQL injections, a check ensures that only data fields in the query are under user control. Wei suggested that a policy ensuring that no two consecutive tokens are tainted would achieve that goal. Similar policies can be used to defeat other types of attacks, such as cross-site scripting, path-traversal attacks, and command injections. These policies are enabled through the availability of fine-grained taint information. Various low-level optimizations allow the performance overhead of the approach to be below 10% for server applications.

Asked whether one could apply the same approach directly to binaries, thus suppressing the requirement that source code be available, Wei explained that it may be possible but would almost certainly preclude the optimizations they used to keep the performance overhead acceptable. Anil Somayaji asked whether hardware support could help improve performance. Wei answered that it was a possibility that had not been fully explored yet.

## INVITED TALK

### ■ *Signaling Vulnerabilities in Wiretapping Systems*

*Matt Blaze, University of Pennsylvania*

*Summarized by Patrick Traynor*

With a recent push for the compliance of VoIP systems with the Communications Assistance for Law Enforcement Act (CALEA), many in the community have focused on the technical hurdles of this proposal. In an attempt to understand the security and reliability of these new eavesdropping systems, Matt Blaze and a team of University of Pennsylvania researchers have examined the inner workings of traditional telecommunications surveillance

technology. As part of the Trustworthy Network Eavesdropping and Countermeasures (TNEC) project, this group explores the challenges behind building both surveillance-friendly and resistant networks. Most important, this work asks the question, "Assuming properly functioning tools, is wiretap evidence trustworthy?"

There are a number of ways in which telecommunications traffic can be intercepted. If direct access to the targeted phone is possible, the eavesdropping party can gain control of the two wires connecting the phone to the network (known as the local loop). Such methods are prone to discovery by the monitored party and are therefore not typically used in law enforcement. This work instead focuses on the use of the loop extender, a wiretapping device placed between the targeted phone and a network switch that allows for monitoring in both traffic analysis ("pen register") and full content modes. The loop extender works by redirecting a copy of the content from the "target line" to recording devices on a "friendly line." Because both traffic analysis and content are always sent to the friendly line, the actual information recorded is set via configuration at the recording device.

Blaze and his team began by examining vulnerabilities in the pen register mode. In order to determine which number a target is calling, the recording equipment decodes the series of unique audio tones sent across the line. In order to accommodate a wide range and quality level of products, both the network switching equipment and eavesdropping tools accept a range of analog tones. The ranges accepted by the devices, however, are not the same. Accordingly, Blaze was able to show that sending tones just outside the range of the telecommunications switch (either above or below) but within the range of the loop extender allowed the party being eavesdropped upon to forge the list of dialed numbers recorded in pen register mode.

Blaze then discussed subtle vulnerabilities in the audio recording mechanism used for eavesdropping. The taping of audio content automatically begins at the cessation of the "C-tone," an audio tone transmitted across a phone line when it is not in use. Accordingly, when the C-tone is again detected, the recording equipment automatically shuts itself off. To prevent sensitive material from being recorded, a targeted party can simply play the C-tone continuously into the receiver. This technique is successful even when the C-tone is played at 1/1000 of its normal volume.

The new CALEA standards fix many of the problems associated with loop extender wiretaps. Eavesdropping now occurs at the switch itself and signaling has been separated onto a separate channel. Unfortunately, many vendors continue to offer systems that are compliant to the C-tone shutoff mechanism. Depending on the configuration of these new CALEA-compliant devices, modern eavesdropping systems may also be susceptible to the vulnerabilities discovered for loop extender systems.

The attendees of the talk asked Matt a wide variety of questions. Because of the illegality of wiretapping equipment, many were curious as to how this group was able to perform their work without fear of legal recourse. Matt mentioned that because this research was part of an NSF grant devoted to wiretapping technology, they had implied consent to carry it out. Others were curious about testing the local loop by the phone companies. Matt said that it was indeed possible for the phone company to detect the deviation of frequencies from the accepted band but that the wide range of quality across user devices means that such divergence from the standard may only be the result of poor construction. Matt's talk concluded with a discussion of the practical difficulties facing IP wiretapping efforts.

## NETWORK SECURITY

*Summarized by Wei Xu*

### ■ *SANE: A Protection Architecture for Enterprise Networks*

*Martin Casado and Tal Garfinkel, Stanford University; Aditya Akella, Carnegie Mellon University; Michael J. Freedman, Dan Boneh, and Nick McKeown, Stanford University*

Martin Casado started his talk by pointing out that traditional techniques for putting access control onto enterprise networks are associated with many problems, such as inflexibility, loss of redundancy, and difficulty in management.

To address these limitations, Martin proposed SANE, a Secure Architecture for the Networked Enterprise. SANE introduces an isolation layer between the network layer and the datalink layer to govern all connectivity within the enterprise. All network entities (such as hosts, switches, and users) in SANE are authenticated. Access to network services is granted in the form of capabilities (encrypted source routes) by a logically centralized server (Domain Controller) according to high-level declarative access control policies. Each capability is checked at every step in the network.

Martin also presented several important details in SANE (such as connectivity to the DC, establishing shared keys, and establishing topology), as well as the countermeasures that SANE offers for attack resistance and containment. He also mentioned that their prototype implementation showed that SANE could be deployed in current networks with only a few modifications.

■ ***PHAS: A Prefix Hijack Alert System***

*Mohit Lad, University of California, Los Angeles; Dan Massey, Colorado State University; Dan Pei, AT&T Labs—Research; Yiguo Wu, University of California, Los Angeles; Beichuan Zhang, University of Arizona; Lixia Zhang, University of California, Los Angeles*

In this talk, Mohit Lad first briefly introduced the BGP (Border Gateway Protocol) and then illustrated the widely reported BGP prefix hijack attack, in which a router originates a route to a prefix but does not provide data delivery to the actual prefix.

After that, Mohit presented a Prefix Hijack Alert System (PHAS). The objective of PHAS is to provide reliable and timely notification to prefix owners when their BGP origin changes, so that prefix owners can quickly and easily detect prefix hijacking events and take prompt action to address the problem. PHAS uses BGP data collectors (especially RouteViews and RIPE) to observe the BGP updates. The updates are then provided to the origin monitor, which detects the origin changes and delivers email notifications to the affected prefix owners through a multipath delivery. According to Mohit, PHAS is lightweight and readily deployable.

■ ***Passive Data Link Layer 802.11 Wireless Device Driver Fingerprinting***

*Jason Franklin, Carnegie Mellon University; Damon McCoy, University of Colorado, Boulder; Parisa Tabriz, University of Illinois, Urbana-Champaign; Vicentiu Neagoe, University of California, Davis; Jamie Van Randwyk, Sandia National Laboratories; Douglas Sicker, University of Colorado, Boulder; Scott Shenker, University of California, Berkeley*

Parisa Tabriz gave the first part of this talk, in which she explained the motivation for their work by arguing that the emerging driver-specific exploits were particularly serious for the 802.11 wireless devices because of their wide deployment and external accessibility. Device driver fingerprinting can be of help for an attacker to launch a driver-specific attack.

Damon McCoy then presented the details of their passive fingerprinting technique, which identifies the wireless device driver running on an IEEE 802.11 compliant device. Their technique is based on the observation that many of the details of the active scanning process are not fully specified in the IEEE 802.11 standard, and hence they are determined by wireless driver authors. As a result, drivers can be distinguished by their unique active scanning patterns. Damon said that they had generated signatures for 17 different wireless drivers, and their evaluation showed that this fingerprinting technique could quickly and accurately fingerprint wireless device drivers in real-world wireless network conditions. Finally, Damon discussed several possible ways to prevent the 802.11 wireless driver fingerprinting.

When asked whether the fingerprint of a wireless driver would change in different operating systems or when the driver interacted with different access points, Damon and Parisa answered that the active scanning behavior should depend only on the driver implementation, but they had not looked into these particular problems.

## INVITED TALK

■ ***Turing Around the Security Problem: Why Does Security Still Suck?***

*Crispin Cowan, SUSE Linux*

*Summarized by Bryan D. Payne*

Crispin Cowan began this invited talk saying that "security sucks" more than any other aspect of computing. He backed up this statement by showing how Turing's theorem can be used to show that security is an undecidable problem. In addition, security is harder than correctness. Although correctness is important, security increases programmer burden, since attackers can produce arbitrary input. History has shown that neither money nor even diligent corporate practices are sufficient to solve the problem.

Crispin explained that one approach to security is to use heuristics. This is seen, for example, in static analyzers. The problem with heuristics is that they are not perfect. Heuristics cannot analyze all programs, and they provide imperfect results in other applications.

As an alternative, Crispin suggests that security professionals use the OODA Loop. Colonel John Boyd (USAF) invented the idea of an OODA Loop. OODA stands for Observation, Orientation, Decision, and Action. The OODA Loop provides a pattern for fighter pilots to use in decision-making. In general, the person with the fastest OODA Loop will win a battle. Crispin spent a large portion of the talk mapping the process of security to the OODA Loop.

The OODA Loop can be mapped to the three critical questions: when, where, what. Crispin walked through many security tools and concepts, mapping each to one of these three questions. He covered topics such as Saltzer and Schroeder's principles of secure design, syntax checkers, semantic checkers, type-safe languages, kernel enhancements, intrusion detection, intrusion prevention, network access controls, host access controls, capabilities, and more.

In closing, Crispin changed gears and spoke about AppArmor, SUSE's approach to improving application security. Crispin described the fundamental difference between AppArmor and SELinux: AppArmor uses path names and SELinux uses labels. He also acknowledged that AppArmor is a work in progress; some types of protection are not currently possible.

In the Q&A session, Pete Loscocco, a leader of the SELinux project, suggested that the differences between SELinux and AppArmor are more fundamental than Crispin stated because SELinux tries to be comprehensive whereas AppArmor has less control. Crispin acknowledged that there are trade-offs and suggested that SELinux achieves higher security but also has a higher cost. Other people asked questions about practical security issues. Why aren't companies doing more about security? Crispin believes that they are secure enough for their purposes. What's being done to address the security problems today? Crispin pointed to better programming languages and various forms of isolation. How does one properly extend security from the kernel into the applications? Crispin suggested using discrete programs that cooperate (e.g., Postfix).

**PANEL**

## *Major Security Blunders of the Past 30 Years*

*Matt Blaze, University of Pennsylvania; Virgil Gligor, University of Maryland; Peter Neumann, SRI International Computer Science Laboratory; Richard Kemmerer, University of California, Santa Barbara*

*Summarized by Madhukar Anand*

Matt Blaze: We are going to have more damage because of excess attention to security. For instance, consider an alarm system. It is designed such that the alarm is triggered as quickly as possible. Consequently, any attack that aims at not setting off the alarm system is entering into a arms race with the alarm sensor design. Although the alarm system could win such an arms race, it must be noted that the main objective of the attacker is not to defeat the sensor system but to break the system. A simple strategy of setting off the alarm repeatedly would do the trick. The police might be tricked into losing faith in the alarm system.

Another example is the signaling in telephone systems. The objective of the attacker here is to defeat the billing system and make free long distance calls or call unauthorized phone numbers. The protocols are designed by people who did not know they were designing a security function. The only people who are motivated to break into such a system are the bad guys.

The Telnet protocol was designed with an option for encryption (DES 56-bit key in a 64-bit package). The newer library, to be more compliant with the DES standard, had 8 bits designated as the checksum. If the checksum would not tally, then the key would be all 0s. Because of this, if there were bit errors, there was a 1/256 chance of using encryption. In 255/256 cases, the key would be all 0s. So, this is an counterexample to the principle "Good code is secure code." Code that checks secure values in this case indeed made it less secure.

Virgil Gligor: Blunder 1: Morris worm—Buffer overflow in fingerd. This was the first tangible exploit of a buffer overflow, the first large-scale infection engine, and the first large-scale DDoS attack. Although there were no lessons learned for a long time, it did result in the creation of CERT. It also led to the realization that any new technology introduces new vulnerability and vulnerability removal takes 6 to 12 years, creating a security gap. Also, we learned that "security is a fundamental concern of secondary importance."

Blunder 2: Multilevel Secure Operating System (MLS-OS). This was pushed by the defense establishments in the United States and Europe. However, there was no market for MLS-OSes as they break off-the-shelf applications and are often hard to use. In fact, the stronger the MLS-OS, the worse the system (e.g., B2 secure Xenix broke all our games). This blunder largely drained most security funding and was a major R&D distraction.

Blunder 3: Failed attempts at fast authenticated encryption (in 1-pass -1 cryptographic primitive.) This blunder was largely academic. Starting with CBC+CRC-32 systems in 1977 to NSA dual counter mode + XOR, each successive system was built only to be broken later. Together, they constitute the largest sequence of cryptographic blunders. The upshot of this is the realization that some of the simplest cryptographic problems are hard. The lesson to be learned is to stick to basic system security research.

Richard Kemmerer: Blunder 1: U.S. export controls of cryptography. These controls were based on international traffic in arms regulations. However, they were very difficult to enforce. The lessons learned from the export control episode are that they make sense but change constantly and that the enforcers have no sense of humor—a case in point being the Verification Assessment study (Kemmerer, 1986), which was published with the warning that its export is restricted by the arms export control act. This hampered the publication of the study and resulted in many copies being stocked and not reaching their readership.

Blunder 2: Kryptonite Evolution 2000 U-Lock. Although the Kryptonite was advertised as the "toughest bike lock," all it took to break it was a Bic pen. After cutting small slits in the end of the pen's barrel to ease it in, the lock opened with a single twist.

Blunder 3: Australian Raw Sewage Dump in March 2000. Vitek Boden from Brisbane, Australia, hacked into a local waste management computer system and directed raw sewage into local rivers, into parks, and even onto the landscaping of a Hyatt Regency hotel. Boden had previously been fired from the company that installed the waste management system, and he vandalized the public waterways as an act of revenge. The lessons learned from this were to pay attention to security in SCADA systems and to insider threats. In conclusion, "Beware of gorillas invading your system while you are counting basketball passes," which was depicted by showing a video where a person in a gorilla suit walks through the scene but is unnoticed by most of the attendees until the second showing of the video.

Peter Neumann: The big picture is that security is complex, it is a end-to-end system emergent property, and there is a weakness in depth. We do not really learn from experience, as some of the blunders keep recurring. Security is holistic. So, we should not consider it in isolation. For instance, application security is easily undermined by OS security.

Propagation Blunders: 1980 ARPANET collapse. The collapse, resulting from a single point failure in one node, ended up bringing down the entire network. In 1990, there was a half-day when AT&T long lines collapsed, again as a result of a single point of failure. Yet another example of propagation blunders are the repeated power outages from 1967 to August 2003. In some sense, we don't design our infrastructures and our systems to withstand things as simple as a single point failure. The standard answer is, "This can never happen."

Backup and Recovery Blunders (e.g., air traffic backup and recovery failures): In July 2006, there was a power outage in Palmdale, CA, including the air traffic control center. The center automatically switched to backup diesel generators. They worked for about an hour but then the system that switches the center between commercial and backup power failed and the building went dark. In 1991, three New York airports were closed as backup batteries were accidentally drained. Other examples of such blunders include the Swedish train reservation system failure and the Japanese stock exchange system failure in November 2005. There have been blunders where there was no backup, such as when the New York Public Library lost all its references.

Software Blunders: There have been many types of buffer overflows. Programming language research could help reduce these, but they could also prove to be a hindrance. In fact, Multics was mostly free of buffer overflow vulnerabilities, owing to the use of PL/I as the implementation language. PL/I required an explicit declaration of the length of all strings.

Election Systems Blunder: The requirements of an electronic voting system include end-to-end reliability, integrity, and accountability. Therefore, Help America Vote (HAVA) was a huge blunder. Because none of the electronic paperless systems were auditable, they lack integrity.

The lessons learned are that individual cases may be less important than the fact that we see the same types of problems. We need a massive cultural change in how we develop software.

More resources on risks and trustworthy architectures can be found at http://www.csl.sri.com/users/neumann/chats4.html.

## INVITED TALK

### ■ *Aspect-Oriented Programming: Radical Research in Modularity*

*Gregor Kiczales, University of British Columbia*

*Summarized by Kevin Butler*

Gregor Kiczales gave a practical, code-oriented talk that provided an introduction to aspect-oriented programming (AOP). Kiczales and his team originated the idea of AOP while he was at PARC. The talk focused on AOP and what it is, what makes it interesting, and how software will be different with AOP compared to what has come before. The focus was on the Aspect/J programming language, a seamless extension to Java that is fully supported by the Eclipse open

source project and is a model for other AOP tools.

One of the biggest benefits of AOP is its ability to aid code expressiveness: The code looks like the design, and what is going on in terms of operations is clear. As an example, Kiczales pointed to a Java program called jHotDraw which allows joining of points and lines, setting colors, and other graphical tasks. This can be designed and implemented with object-oriented programming, where shapes become classes. There are also objects relating to the display of the shapes and to perform update signaling (e.g., when shapes change). The goals for any programming paradigm are expressiveness, modularity, and abstraction. For jHotDraw, object-oriented programming allows good support of these goals for shapes and their display, but it is not good for update signaling. The structure of signaling is not localized, clear, or declarative, and the resulting solution is neither modular nor abstract. Signaling is clearly not localized as it cuts across multiple objects. With AOP, it is possible to provide the three goals set out.

Some of the terminology specific to AOP includes the idea of *join points*, which are defined points in a program flow (Aspect/J allows for dynamic join points) and *pointcuts*, a set of join points on which a predicate is based that may or may not match. Pointcuts are composed like predicates, and a set of primitive pointcuts are defined in Aspect/J. When a pointcut is reached, a specified piece of code, called *advice*, is run. To go back to the drawing example, there is an observer pattern aspect of the system, and some points in the system's execution are changed; after returning for a change, the display is updated. Updated calls would be scattered and tangled with object-oriented programming; by contrast, with AOP, values at the join points are defined, a pointcut can explicitly expose certain values, and advice can use these explicitly exposed values. The key to AOP is its ability to deal with *crosscutting* across multiple concerns. For example, a pointcut in a model can specify a slice of a different part of the world to provide an interface. This could be particularly pertinent to industry: One can walk up to a big system, drop pointcuts in it, and program against the system even if it doesn't have the structure the programmer was expecting. In response to questions about this, Kiczales explained that one of the fundamental differences with AOP is that there is no need to explicitly signal events as is necessary with OOP. For example, one could take a million lines of already written code and do pointcuts without even having the source code for the original system. When asked whether the goal of AOP was to be able to retrofit features into legacy code, Kiczales was circumspect and careful in his answer. The biggest goal of AOP is to modularize crosscutting concerns, which sometimes can be done for legacy code and sometimes not. Oftentimes, new features are crosscutting concerns.

Kiczales expressed some thoughts as to whether going further than Aspect/J could be feasible. Referencing the book *On the Origin of Objects* by Ryan Smith (where objects refer to actual real-world things, not class instances), he talked about perception and how we see the world in different ways (e.g., a glass half empty vs. half full), and how do we both grab the same thing. Registration is the process of parsing objects out of the fog of undifferentiated "stuff," and we are constantly registering and reregistering the world. We also possess the ability to process critically: We have multiple routes of reference, such as the ability to exceed causal reach (e.g., describing someone as closest to average height in Gorbachev's office) or indexical reference (e.g., the one in front of him); allowing this sort of expressiveness into programming could be similarly useful. Join point models can decompose software into different ways and atomize into the fog of undifferentiated points, with connections and effects made through registration. All could have causal access to the same point but access it in different ways, given the multiple routes to reference.

Much of the Q&A session focused on the security aspects of AOP. For example, if AOP or Aspect/J can touch a JAR file, then all bets are off. Audit logging is a clear example of where AOP is useful, but what other security uses are there? Is ACL checking a potentially good use? Kiczales responded that his group intentionally did not touch on the uses for security, leaving that to security professionals, but intimated that there were certainly security problems that could best be attacked with AOP. Gary McGraw commented that this logic could be taken only so far: If something is a security feature, it can probably be done as an aspect, but security is not just a bunch of features. Other questions included dealing with aspect clashes, to which the best defense is good software engineering and a quality codebase; how to debug code, which has improved greatly as Aspect/J has matured; and the problems of retrofitting legacy code with access control hooks, which is nontrivial. Kiczales

responded to this by commenting that no sane person claims that one should be able to take a system and add modular implementations of crosscutting without having to jigger the source code a little bit, particularly when dealing with the domain functionality of a system. However, refactoring and some mild editing of the code could make all the difference.

## STATIC ANALYSIS FOR SECURITY

*Summarized by Lionel Litty*

### ■ *Static Detection of Security Vulnerabilities in Scripting Languages*

*Yichen Xie and Alex Aiken, Stanford University*

Alex Aiken observed that Web applications are increasingly popular and that they present application-level vulnerabilities that are hard to prevent using low-level security mechanisms. Instead, static analysis may prove valuable when hunting for security bugs in such applications. He reported that his group was successful in performing static analysis of a scripting language (PHP) to find SQL injection vulnerabilities, despite the difficulties inherent in analyzing scripting languages: dynamic typing, implicit casts, weak scoping, and extensive use of string manipulation functions and hash tables.

To scale to large applications without making the analysis too imprecise to detect bugs reliably, three levels of abstraction were used: intrablock level, intraprocedural level, and interprocedural level. The result of the symbolic simulation of each code block is summarized before performing the analysis of each procedure. The result of this analysis is in turn summarized to perform interprocedural analysis. Each summary captures exactly the information necessary to perform the next step of the analysis, which is the key to scaling. Applying this technique to six large applications resulted in the discovery of 105 confirmed vulnerabilities with only a small number of spurious warnings.

Christopher Kruegel asked how the tool dealt with variable aliasing. Alex answered that the tool did not perform sound tracking of aliasing outside of the block level and that this was one way in which the tool was not a verification tool, meaning that it will not guarantee the absence of vulnerabilities. He also highlighted that programmers often do not use the full power of a programming language. This enables the tool to perform well despite making a number of simplifications.

### ■ *Rule-Based Static Analysis of Network Protocol Implementations*

*Octavian Udrea, Cristian Lumezanu, and Jeffrey S. Foster, University of Maryland*

Octavian Udrea argued that whereas formal methods have been used extensively to check network protocol designs, the actual implementations of these protocols often do not get the same level of attention. He described a tool called Pistachio that addresses this situation by focusing on static verification of a protocol implementation against a high-level specification of the protocol. This manually constructed specification, derived from documentation describing the protocol, consists of a set of rules that the protocol must adhere to.

With the help of a theorem prover, execution of the implementation is then simulated. Developing rules for the SSH and RCP protocols took the authors only seven hours, although these rules did not cover all aspects of the protocols. Implementations of these protocols were then analyzed, and rule violations were found in the LSH implementation of SSH and the Cygwin version of RCP. These violations were checked against a database of known bugs. Of the warnings produced by Pistachio, 38% were spurious. Pistachio failed to find 5% of known bugs. More information is available at http://www.cs.umd.edu/~udrea/Pistachio.html.

David Wagner asked how the authors were able to estimate the false-negative rate. Octavian explained that this number corresponded to known implementation bugs that Pistachio did not find. This number might be inflated by still unknown bugs in the implementation.

### ■ *Evaluating SFI for a CISC Architecture*

*Stephen McCamant, Massachusetts Institute of Technology; Greg Morrisett, Harvard University*

***Awarded Best Paper***

Software-based Fault Isolation (SFI) is a sandboxing technique used to confine the effects of running a piece of code from the rest of the system by inserting checks before every memory write and every jump. While the technique has been shown to work for RISC architectures, Stephen McCamant showed that it could be applied to a CISC architecture, the IA-32 architecture, as well. The key challenges in building a prototype, PittSFIeld, were the variable length of the IA-32 instructions and their lack of alignment constraints, potentially allowing complex code obfuscation.

These challenges were addressed by forcing instruction alignment via assembly code rewriting and by adding padding. In addition,

the authors implemented new optimization techniques and carefully analyzed the performance overhead introduced by SFI, which is a 21% slowdown on the SPEC benchmark. One reason for the slowdown is increased cache pressure caused by the 75% increase in size of the binary code as a result of padding. Finally, Stephen emphasized that the security of PittSFIeld relies on a small verifier. The authors formally proved, using the ACL2 theorem proving system, that the verifier checks ensure confinement. More details can be found at http://pag.csail.mit.edu/~smcc/projects/pittsfield.

The audience asked whether the same technique could be performed directly on the assembly code. Stephen answered that it might be possible with a very good disassembler or if additional symbol information is available. Stephen was also asked the difference between SFI and Control Flow Integrity (CFI). He said that the two approaches were very similar. SFI provides memory isolation, but this can be added to CFI as well. Timothy Fraser mentioned that he knew of failed efforts to implement SFI for IA-32 and commended the authors.

## INVITED TALK

### *Surviving Moore's Law: Security, AI, and Last Mover Advantage*

*Paul Kocher, Cryptography Research*

*Summarized by Micah Sherr*

Paul Kocher's talk was divided into two parts. In the first half, he explored how complexity can be used to secure systems. He began by examining the effect of Moore's Law on cryptography. On the one hand, increased CPU performance seems to favor the cryptographer. For example, moving from DES to 3DES requires only three times the CPU cost, at the same time requiring an exponential increase in work by the cryptanalyst. However, as Paul points out, systems are becoming more complex, and added complexity (and lines of source code) raises the number of potential vulnerabilities dramatically. The growth of human intelligence does not necessarily match the growth of system complexity. There is therefore a need to harness complexity to improve security rather than hinder it.

Paul proposes increasing complexity by adding components to systems and connecting those components in a stream. The output of the system is the combination (e.g., XOR) of all of the components' outputs. If one system fails or is compromised, the overall security of the system does not necessarily falter. Generally, he advocates adding "micro CPUs"—independent and isolated execution areas that are specialized for different operations. The micro CPUs communicate with each other to produce the final result.

In the second half of the talk, Paul described the current state of security as a tug-of-war between those who protect systems and those who attack them. Paul believes that companies no longer believe that security is binary and that they have adopted the strategy of minimizing (rather than preventing) the amount of harm done to their systems. The major defense mechanism is the patch, which is applied after an attack, but which hopefully reduces the possibility of further attack. On the flip side, the attacker attempts to exploit new and unpatched vulnerabilities. This leads to a stalemate in which attacks are closely followed by fixes. According to Paul, this stalemate is the best-case scenario for the defender: Playing for a stalemate requires less devotion than comprehensive security fixes, stalemates allow defenders to place the blame on others (those who haven't applied the patches), and developing patches is easier than in-house security testing. However, playing for a stalemate leaves the defender with nonoptimal security.

Paul concludes by considering the future of this tug-of-war between defender and attacker. To some degree, generation of attacks (exploits) and defenses (patches) can be automated. At the same time, the undecidability of certain problems means that there are limits to what automation can accomplish. To some degree, the human element will always be involved in the automation process.

## INTRUSION DETECTION

*Summarized by Michael Locasto*

### *SigFree: A Signature-Free Buffer Overflow Attack Blocker*

*Xinran Wang, Chi-Chun Pan, Peng Liu, and Sencun Zhu, The Pennsylvania State University*

Xinran Wang began by illustrating the problems with current approaches to blocking buffer-overflow based attacks. Such attacks can be previously unseen or potentially delivered by polymorphic malware. Remote buffer overflows are typically driven by network input containing the exploit code, and Xinran and his coauthor's key insight is that detecting the presence of legal binary code in network flows was one way to recognize such attacks without having to resort to a signature-based scheme.

Xinran described two techniques their Instruction Sequence Analyzer (ISA) uses to build an extended instruction flow graph (EIFG), a construct similar to a control flow graph. Since x86 is a

very dense instruction set, instruction sequences can be derived from most binary strings, and the authors propose building an EIFG from the network traffic. Intuitively, the larger the EIFG, the more likely it is that a certain binary string is a working x86 program. The first technique for building an EIFG employs pattern detection to provide a frame of reference for the ISA—it detects the push-call instruction sequences that represent a system call.

Xinran outlined a more sophisticated method (Data Flow Anomaly), which is based on detecting abnormal operations on variables. The motivation for this technique is that a random instruction sequence is full of these types of data flow anomalies, but a useful x86 program (such as one provided by an attacker) is not. This latter technique is also more resistant to polymorphic attacks, since the malware decoder does not necessarily need to make system calls, but still must include useful instructions. The authors report on fairly good results: No false positives were observed in a "normal" test set, and their techniques detected about 250 attacks and variations. They are looking forward to using a weighted scheme to frustrate attackers who may know the threshold of useful instructions for the DFA scheme. A follow-up question on how well this technique may apply beyond the x86 instruction set was tabled for offline discussion. An attendee from Cisco asked whether the code for the system would be publicly available, since this type of capability was of widespread interest, but Xinran indicated that the techniques were currently under patent review. Finally, another attendee from MIT asked about the nature of the "normal" requests that actually contained fairly long sequences of code. That discussion was taken offline.

### ■ *Polymorphic Blending Attacks*

*Prahlad Fogla, Monirul Sharif, Roberto Perdisci, Oleg Kolesnikov, and Wenke Lee, Georgia Institute of Technology*

Prahlad discussed the design and implementation of a class of malware capable of dynamically adapting its payload to the characteristics of the surrounding network traffic in order to bypass content-based anomaly sensors such as PayL. The key idea is that the malware is able to sneak past the anomaly detector because purely statistical content anomaly detectors discard both the syntax and the semantics of the normal traffic content.

The malware has two capabilities that allow it to create a variant that will not trip the content anomaly detector. First, it is able to observe a portion of the target system's network traffic. Second, it knows the algorithm that the anomaly detector uses to construct a model. After creating an initial and approximate model of the traffic that the target system is receiving, the malware creates a variant of itself using shellcode encryption and padding to match the generated profile within some error bound. In their experiments using PayL as the content anomaly detector, the authors found that all they needed was 20 to 30 packets to learn the target profile. The authors closed by suggesting some possible countermeasures to the polymorphic blending attack, including the higher-cost techniques of actually parsing the packet data rather than using solely statistical methods. It may also be possible to use multiple independent models or to somehow randomize the implementation of the anomaly detector algorithm. At the end of the presentation, the attendee from Cisco reprised his question on the availability of the system, and Prahlad answered that they shared code with some partners of the lab. A second questioner asked for clarification as to the practicality of the first countermeasure (doing deep packet inspection). Prahlad answered that with appropriate support or in some types of low-traffic environments, such inspection would not be prohibitively expensive.

### ■ *Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection*

*Holger Dreger, Anja Feldmann, and Michael Mai, TU Munchen; Vern Paxson, ICSI/LBNL; Robin Sommer, ICSI*

Holger raised the question of how network intrusion detection systems actually know which protocol decoder should be applied to the traffic that these systems observe. Most current systems simply rely on the transport layer port numbers in order to make this decision. This assumption does not hold for standard services that are run on nonstandard ports nor for standard or nonstandard services that are run on the "incorrect" port.

Holger then used some summary statistics of a large body of data to further explicate the problem. The data was a full packet capture over a 24-hour period, totaling some 3.2 TB, 137 million TCP connections, and 6.3 billion packets. The authors ran the Linux netfilter i7 protocol signatures on the traffic trace and focused on HTTP, IRC, FTP, and SMTP. They found that i7's matching was good for the normal case, that is, when a particular protocol appeared on a port it normally does. But a significant portion of the traffic defied classification or was incorrectly classified.

The key question the authors seek to address is the problem of distinguishing among network services without taking a hint from the port number. Holger presented the design and implementation of a modification to the Bro NIDS that performs dynamic analysis of the observed traffic: The system matches multiple possible protocols in parallel. The system provides the mechanism for doing so; whether or not to commit to a particular parsing decision is left as a matter of thresholding and policy.

Holger described a number of interesting results, including that most connections on nonstandard ports in their traffic trace were HTTP and that the protocols tunneled therein were mostly P2P services, some raw HTTP, and some FTP traffic. They were able to identify and close some open SMTP relays and HTTP servers that violated the site's security policy and have been able to detect some botnets at one of the author's sites. Holger closed by indicating that the system will be incorporated into Bro.

The first question focused on how difficult it would be to add a new protocol definition to the system. Holger indicated that it was a moderate amount of work, but Bro provides a great deal of infrastructure for doing this already. The remaining questions focused on the system's ability to handle tunneled traffic in various forms, in particular, how easy it would be for an attacker to craft input that prevented the parallel logic from making a choice between two protocols. Holger replied that the system provides the mechanism, and the policy to control it is up to the user or site administrator. Even so, he indicated that the combination of protocol signature matching and dynamic analysis somewhat alleviated this problem.

### ■ *Behavior-Based Spyware Detection*

*Egin Kirda and Christopher Kruegel, Technical University Vienna; Greg Banks, Giovanni Vigna, and Richard A. Kemmerer, University of California, Santa Barbara*

Egin gave an engaging talk about the techniques used in real spyware systems and described the author's system for detecting this spyware. He began by pointing out that spyware is a burgeoning problem, and that it is difficult to identify this type of malware because it often comes in many different guises and is potentially installed by a trusted but unwary user. Signature-based spyware solutions are often on the losing end of an arms race.

However, spyware often cannot escape its main goal, and that goal is to gather information about a user's behavior and then transmit that information somewhere. This type of anomaly or behavior-based detection is a promising technique for detecting spyware that doesn't yet have a signature.

The authors focus on spyware for Internet Explorer, and Egin said that most spyware registers as a Browser Helper Object (BHO) or toolbar (for all practical purposes, a toolbar is no different from a BHO). One major reason for BHO's popularity as a spyware vehicle is that high-level user behavior and data are readily available to a BHO, thus greatly simplifying the implementation of the spyware component: It can simply use the events, data objects, and services that IE provides to all BHOs. Egin indicated that their techniques are of limited applicability for spyware that does not use COM services. For example, spyware may utilize other forms of communication such as issuing GET requests to URLs that are under the attacker's control.

Egin described their implementation of a "stub" IE instance that intercepts communication between the browser and BHOs. The system combines static and dynamic analysis to drive the training phase. The dynamic analysis records all "interesting" COM calls and provides the starting point for static analysis to construct a control flow graph to see whether information could have been leaked. For all of their 51 test samples (33 malicious/spyware, 18 benign), the false-negative rate was zero, but the false-positive rate only improves as the system moves toward the combination of static and dynamic analysis. One interesting result is that the privacy/P3P plugin acts like spyware, according to the authors' behavior characteristic: It reads a URL and then contacts the Web site. After the talk, an attendee asked whether or not it would be possible for a malicious BHO to detect that it was being run by the detector version of the browser and adjust its behavior accordingly. Egin said that this situation was similar to a program detecting whether it was running in a VM, and that the countermeasures taken by the spyware may be detectable, but it was certainly an avenue for future research.

## INVITED TALK

### ■ *DRM Wars: The Next Generation*

*Ed Felten, Princeton University*

*Summarized by Tanya Bragin*

For a number of years digital rights management (DRM) has been a hot topic in the press and a focus of many research and

development efforts. But has there been any significant progress since the Digital Millennium Copyright Act (DMCA) of 1998 bolstered DRM by criminalizing any attempts to circumvent it? And after the public outrage over the Sony rootkit fiasco, what future awaits DRM? Ed Felten is uniquely positioned to answer these questions. He spent many years as a computer scientist in both industry and academia, but in recent years he has been increasingly focusing on legal and policy issues related to technology.

"'Rights Management' is somewhat of an Orwellian term," stated Ed Felten. "It implies that DRM advocates want to manage your rights, not control what you do." But in reality DRM software by the very nature of what it tries to accomplish must restrict users' ability to control their personal computers in order to prevent them from copying restricted content. When DMCA came out, it was considered reasonable for users to surrender some control in order to protect the artists' right to collect revenue for their work. However, it is unclear that technology can deliver on the DRM promise, so some people are starting to question why users have to hand over control of their machines to DRM software companies.

According to Ed, the main issue with the assumption that technology can actually solve the problem of protecting digital content is the "rip once, infringe everywhere" phenomenon, published in what became known as the "Darknet Paper" by Microsoft Research in 2002 (www.freedom-to-tinker.com/?p=206). Given the ease with which people can participate in peer-to-peer file sharing networks, it is only necessary for one person to defeat DRM in order for everyone else to have ready access to the unprotected content. This type of threat model is really hard to defeat and DRM advocates have gradually become convinced that this problem is intractable in practice.

With this realization, incentives for deploying DRM software have also changed. Originally DRM was supposed to provide antipiracy features that benefited artists and ensured compliance of digital content with the copyright law. However, given the apparent infeasibility of these goals, DRM software has come to serve other purposes. First, it allowed digital content distributors to perform *price discrimination*, which is enabled by DRM's ability to hinder transfer of content from user to user. Second, it allowed some organizations, such as Apple, to gain significant market share through *platform locking*. Apple's successful combination of iPod, iTunes, and the Apple Store has created an unrivaled digital music sale franchise.

It is unclear that these new rationales for using DRM continue to benefit artists. For example, Apple, publicly published an "interoperability workaround" that allows users to burn protected content to a CD and then rip that content from the CD in an unprotected format. Similarly, price discrimination primarily benefits digital content distributors, because it allows them to make a profit on products they otherwise could not afford to produce. Although price discrimination can also benefit artists and consumers, since it allows certain types of products to come to market, arguably it can be achieved using techniques other than DRM.

Ed expects to see more efforts by companies to use DRM for self-serving purposes, while continuing to claim that antipiracy is the ultimate goal. Ed warned audience members to look out for these troubling developments, since they are likely to hinder interoperability, complicate products, and frustrate consumers. In Ed's opinion, DRM policy should be neutral, neither discouraging nor bolstering its use. Given the "inherent clumsiness of DRM," he believes that market forces would ultimately decide against it. For more information about DRM and other legal and policy issues related to technology, visit Ed Felten's blog, "Freedom to Tinker," at www.freedom-to-tinker.com.

## SYSTEM ASSURANCE

*Summarized by Bryan D. Payne*

### *An Architecture for Specification-Based Detection of Semantic Integrity Violations in Kernel Dynamic Data*

*Nick L. Petroni, Jr., and Timothy Fraser, University of Maryland; AAron Walters, Purdue University; William A. Arbaugh, University of Maryland*

Current systems design makes the kernel a high-value target for attackers. Defenses such as load-time attestation and run-time attestation are valuable, but these are only capable of detecting changes to static data. Nick Petroni described how attacks can exploit this limitation to avoid detection, using an example of manipulating dynamic data structures to hide a process in Linux.

The proposed solution is to create a high-level model of the dynamic data and to use this model to validate the data. This approach, which was inspired by work from Demsky and Rinard, uses a data structure specification language to model constraints on the data. Nick showed a simple example that would catch the previously mentioned attack. The technique has

some limitations, including the possibility of missing short-lived changes and the need for a kernel expert to model the constraints.

In the Q&A session, the session chair asked whether people would actually use a parallel language such as this. Nick believes that they would and that companies could provide the specification for binary operating systems such as Windows or Red Hat Linux.

■ ***vTPM: Virtualizing the Trusted Platform Module***

*Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn, IBM T.J. Watson Research Center*

Stefan Berger presented an architecture for creating virtualized trusted platform modules (TPMs). Multiple virtual machines cannot share a single hardware TPM, because of limited resources in the TPM and because the TPM has a single owner. In addition, virtual machines can migrate, whereas a TPM is physically tied to a single machine. The problem is that hardware-rooted security is still desirable.

As a solution to this problem, Stefan presented a virtualized TPM (vTPM) that is linked to the TPM through signed certificates. The vTPM is implemented as a software component, but one could also build it into a secure co-processor for improved security properties. Although there are still some challenges related to virtual machine migration, Stefan explained how the vTPM architecture allows a TPM to be extended to a virtual environment today.

In the Q&A session, the first question involved hardware requirements. Stefan said that TPM is a standard and is available from a variety of vendors. To a question about the size of the vTPM implementation, Stefan answered that it is about 250 kbytes per vTPM.

■ ***Designing Voting Machines for Verification***

*Naveen Sastry, University of California, Berkeley; Tadayoshi Kohno, University of California, San Diego; David Wagner, University of California, Berkeley*

Naveen Sastry presented techniques for building direct recording electronic (DRE) voting machines for easier verification. Current-generation DRE voting machines are built as one large, monolithic system. However, by building these systems using distinct components with minimal data passing, each component becomes easier to verify. Naveen proposed building separate components for each of the security-critical functions. In addition, Naveen recommended rebooting between voters to ensure that each voter starts with a known good system state.

In the presentation, Naveen discussed the need to select specific security properties and design the system to satisfy these properties. Specifically, he looked at two security properties: (1) none of the voter's interactions with a DRE may affect a later voter's sessions, and (2) a ballot may be stored only with the voter's consent to cast. Compartmentalized implementation techniques were used to demonstrate a system that upholds these properties.

In the Q&A session, one questioner asked about the problem of malicious developers. Naveen answered that this can be viewed as another security property and a system designed to handle the problem. Naveen also addressed concerns that a reboot would not clear the system memory by suggesting that one cut power to the system.

## WORK-IN-PROGRESS REPORTS

*Summarized by Manigandan Radhakrishnan*

■ ***Exploiting MMS Vulnerabilities to Stealthily Exhaust a Mobile Phone's Battery***

*Radmilo Racic, Denys Ma, and Hao Chen, University of California at Davis*

This attack is aimed at draining the battery of the victim's cell phone without his or her knowledge. The cell phone discharges the most when it is in the *Ready* state as opposed to *Idle* and *Standby*. The attack works by keeping the cell phone at the Ready state as long as possible even when no useful function is performed. The authors achieve this feat by exploiting vulnerabilities in the Multimedia Messaging Service (MMS), specifically the Packet Data Protocol (PDP) context retention and paging channels. The attack works in two phases. The first phase involves the creation of a *hit list*, which is a list of mobile devices that includes their cellular number, IP address, and model number. The second phase is the battery-draining phase, where repeated UDP/TCP ACK packets are sent to the mobile phone either just before the timer expires or just after the timer expires (so that the network has to page the mobile device). The timer here is the duration the mobile device waits before going to Standby state from the Ready state. The experimental results show that the authors have been able to discharge cell phone batteries at a rate that is up to 22 times faster than that in the Standby state.

■ ***Applying Machine-Model-Based Countermeasure Design to Improve Protection Against Code Injection Attacks***

*Yves Younan, Frank Piessens, and Wouter Joosen, Katholieke Universiteit Leuven, Belgium*

Code injection attacks are more prevalent today, and the authors point out that the countermeasures are developed in an ad hoc manner. The authors suggest a more methodical approach to countermeasure development. They present two approaches, machine-model and meta-model. The machine-model is the model of the execution environment of the program including the memory locations (stack, global tables, etc.) along with the operations performed on them. The relations between the different components and their interactions are captured using UML (although the authors do note that they are working on a better alternative). Such a model not only allows the designer of countermeasures to function at an abstract level but also provides a means to compare and evaluate different countermeasures. The shortcoming of this approach is the fact that the machine model is closely tied to a particular architecture. To overcome this the authors suggest the meta-model, which is an abstraction of several machine models, and which, according to them, provides uniformity when constructing machine models and allows a designer to work out the global principles of a countermeasure independent of a specific platform.

■ ***Building a Trusted Network Connect Evaluation Testbed***

*Jesus Molina, Fujitsu Laboratories of America*

The Trusted Network Connect (TNC), a part of the Trusted Computing Group (TCG) protocols, comprises a set of standards that ensure multivendor interoperability across a wide variety of endpoints, network technologies, and policies. The particular issue that this presentation tries to address is the composition of secure applications developed by different vendors. The author is building a TNC testbed to test various vendors' products under a variety of policies. The idea is to expose any possible vulnerabilities.

■ ***The SAAM Project at UBC***

*Konstantin Beznosov, Jason Crampton, and Wing Leung, University of British Columbia*

Every authorization system has a policy-based decision maker and an enforcement engine. In the model presented in this WiP, they both communicate through message exchange to decide whether a particular authorization request is allowed. If this decision maker fails or is unresponsive, the system either becomes unavailable as no operations are allowed or gets breached because every operation is allowed. To prevent such a situation, the authors propose a Secondary and Approximate Authorization Model (SAAM). SAAM derives approximate authorization decisions based on cached primary authorization decisions. The efficiency of a SAAM system depends on the authorization policy of the system. The authors tested their scheme on a system implementing the Bell-LaPadula model and found a 30% increase in authorization requests that can be responded to without consulting the primary policy decision maker.

■ ***ID-SAVE: Incrementally Deployable Source Address Validity Enforcement***

*Toby Ehrenkranz, University of Oregon*

Routers deployed on the Internet today do not track the source of an IP packet. They base their routing decisions only on the destination address of each packet. The authors contend that this has been the root cause of IP spoofing attacks over the Internet and of the unreliability of the RPF protocol. ID-SAVE works by building "Incoming Tables" for valid IP addresses that can route packets through this router. A packet not matching a valid entry in the Incoming Table is dropped. The authors do reference Li et al.'s work ["Source Address Validity Enforcement (SAVE) Protocol," 2002] and mention that their approach has similar ideas to SAVE. The novelty of ID-SAVE lies in its approach toward deployment of the protocol in the Internet, which has many legacy routers that may not support Incoming Tables. They use a variety of techniques such as packet marking, neighbor discovery, on-demand updates, blacklists, and packet-driven pushback to derive benefits even with partial deployment.

■ ***Automatic Repair Validation***

*Michael E. Locasto, Matthew Burnside, and Angelos D. Keromytis, Columbia University*

Automated intrusion prevention and self-healing software leave the system administrator a little perplexed, since it is just not possible for him or her to manually ensure that the fix actually fixes the vulnerability. To address the issue and motivate further research in this direction, the

authors propose bloodhound, a system that facilitates automatic verification of fixes by subjecting them to the original input that caused the intrusion. This can be achieved by automatically logging suspicious network traffic and reusing the data to test the fixed software. As was pointed out by the authors, the challenges to this approach lie in sifting through enormous amounts of network traffic to identify malicious inputs, in indexing identified input and storing them, and in ensuring that the replay is an effective guarantee to show that the problem is fixed. Also, this technique has time and space limits and has to be optimized for both if this technique is to be efficient.

■ ***Secure Software Updates: Not Really***

*Kevin Fu, Anthony Bellissimo, and John Burgess, University of Massachusetts at Amherst*

Software updates are used by a wide variety of applications to fix bugs and patch security vulnerabilities. This WiP took a comprehensive look at the update mechanisms found in some of the most common applications (Mozilla Web browsers, Adobe Acrobat, etc.). They found these mechanisms to be heavily dependent on the presence of secure networks for correct functioning. And there were instances of them being vulnerable to man-in-the-middle attacks. The software update mechanisms were also found to be prevalent in embedded devices (typically with limited computing power and battery life) that either operate over insecure networks or cannot support secure computations. The specific examples presented were those in use in automobiles, electronic voting machines, and some implanted medical devices. The statement "Help! My heart is infected and is launching a DDoS on my pancreas" to emphasize the need for secure content distribution was not only hilarious but also thought-provoking.

■ ***Integrated Phishing Defenses***

*Jeff Shirley and David Evans, University of Virginia*

This WiP presented an integrated approach to identifying and thwarting phishing attacks. The proposed system has two parts: (1) to identify suspicious emails and Web sites and (2) to prevent users from accessing these phishing sites. This integrated approach to identification combines existing identification heuristics (link obfuscation and email text contexts) with the authors' own idea of gathering URL popularity measures (derived using such common search engines as MSN and Google). The novelty of their system lies in their use of an HTTP proxy that diverts a user to a warning page when he or she tries to access a URL classified as a phishing site. The authors present experimental evidence that this greatly reduces the false-positive and false-negative rates. They also show that this scheme is successful in preventing a user from reaching a phishing site 94% of the time (with an error rate of about 3%).

■ ***The Utility vs. Strength Tradeoff: Anonymization for Log Sharing***

*Kiran Lakkaraju, National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign*

Logs contain a variety of information and some of these may be sensitive information about an organization. To facilitate sharing of logs among organizations, these logs need to be sufficiently anonymized to prevent any leakages. This WiP talked about the FLAIM framework for log anonymization. The authors described a "strength vs. utility" trade-off to decide on the extent of anonymization. Here "utility" refers to the amount of usable information that is still present in the anonymized log, and "strength" refers to the extent of anonymization. The noteworthy fact here is that the stronger the anonymization, the less useful the log is, and vice versa. The FLAIM anonymization framework uses multilevel policies coupled with a library of anonymization algorithms to anonymize logs. The anonymization policy is represented in a XML-based policy language.

■ ***Malware Prevalence in the KaZaA File-Sharing Network***

*Jaeyeon Jung, CSAIL, Massachusetts Institute of Technology*

More and more viruses are reported to use peer-to-peer (P2P) file-sharing networks such as KaZaA to propagate. These malware programs disguise themselves as files (e.g., Winzip.exe or ICQ.exe) which are frequently exchanged over P2P networks; they infect the user's host when downloaded and opened. They leave copies of themselves in the user's sharing folder for further propagation. The authors present a crawler-based malware detector, called "Krawler," built specific for the KaZaA network. The crawler has a signature database and looks for files with different file names but matching signatures known to the crawler. The crawler uses longest common substring to minimize false positives. Based on their experiments, the author reports that they found 15% of crawled files were infected by 52 different viruses, 12% of KaZaA hosts were infected, and 70% of infected hosts were in the DNS blacklists.

■ ***Election Audits***

*Arel Cordero and David Wagner, University of California, Berkeley; David Dill, Stanford University*

Random election audits are a way of ensuring that the election process is fair. When done correctly these audits can provide objective and measurable confidence about the election process. The difficulty lies in the selection of the samples because (1) the process for selection of the samples should be transparent and yet (2) the samples should be as random as possible. For example, the use of software for random selection may produce a random sample but is not a transparent process. To make the selection process fully transparent and to allow public oversight the authors suggest the use of a die (a 10-faced one, to increase the bandwidth). They are wary that the choice of dice may and probably will encounter resistance in adoption because of the public perception that dice are associated with gambling. And they also note that such public perception has led to superior selection techniques having been rejected in the past. They advocate education as the means to shift perception toward acceptance of physical means such as dice and a lottery in use of random selection.

■ ***The Joe-E Subset of Java***

*Adrian Mettler and David Wagner, University of California, Berkeley*

Every process executes with the same set of privileges inside the Java Virtual Machine (JVM). This means that the principle of least authority is not enforced to ensure that each process has just the necessary set of privileges. Joe-E is a subset of Java designed to build secure systems. Joe-E uses capabilities for the enforcement of protection; hence the program can perform only those operations for which it has capabilities. Joe-E has the advantage of permitting safely extensible applications, where an application's privileges can be limited to the capabilities granted, thereby simplifying analysis.

■ ***Prerendered User Interfaces for Higher-Assurance Electronic Voting***

*Ka-Ping Yee, David Wagner, and Marti Hearst, University of California, Berkeley; Steven M. Bellovin, Columbia University*

It is critical that the software present in voting machines be validated. The authors contend that the codebase in commercially existing voting machines is not only huge (37,000 lines) but also contains a lot of code that is related to the user interface (14,000 lines). Prerendering, to generate the user interface before the election, dramatically reduces the amount of security-critical code in the machine, thus reducing the amount and difficulty of software verification required to ensure the correctness of the election result. Moreover, publishing of the prerendered user interface as a separate artifact enables public participation in the review, verification, usability testing, and accessibility testing of the ballot.

■ ***Fine-Grained Secure Localization for 802.11 Networks***

*Patrick Traynor, Patrick McDaniel, and Thomas La Porta, The Pennsylvania State University; Farooq Anjum and Byungsuk Kim, Telcordia Technologies*

This WiP presented a novel approach to ascertaining a user's location in an unforgeable manner. This is essential when the user's location is a necessary part of user authentication. The authors use a network of access points to transmit cryptographic tokens at various power levels. This location information is further refined by using a low-cost radio in the desktop machines present in the vicinity of the location ascertained by the access points. The use of hash functions (as part of the tokens) ensures that this method is resilient against spoofing attacks.

■ ***KernelSecNet***

*Manigandan Radhakrishnan and Jon A. Solworth, University of Illinois at Chicago*

In most of today's operating systems, networking authorizations are practically nonexistent; that is, common networking operations are unprivileged and end-to-end user authentication is done (if at all) on a per-application basis and is never tied to the authorization system of the operating system. The KernelSecNet project is aimed at providing a unified networking and distributed system abstraction that is implemented at the operating-system level, allows networking policy specification, and provides automatic encryption of all traffic between hosts. The authors intend to develop a model that is specific to the KernelSec project and another independent UNIX-based model.

■ ***Taking Malware Detection to the Next Level (Down)***

*Adrienne Felt, Nathanael Paul, David Evans, and Sudhanva Gurumurthi, University of Virginia*

The WiP presented a novel technique for malware detection using the disk processors. The detection works by having the disk processor monitor sequences of I/O requests and identify sequences that correspond to known malicious actions (essentially, a signature). Implementing the detection at this low a layer makes the mechanism immune to most subversion mechanisms that are present in the layers above and makes it extremely hard to circumvent. Also, this mechanism

works in isolation from the operating system and can operate even when the host is compromised. The authors mention that this technique is more effective when the processors can store more meta information about a request, for example, whether it is a create-file or a remove-file request.

■ *Data Sandboxing for Confidentiality*

*Tejas Khatiwala, Raj Swaminathan, and V. N. Venkatakrishnan, University of Illinois at Chicago*

The authors present a technique they call "Data Sandboxing," which allows information flow confidentiality policies to be enforced on different parts of a legacy application to prevent leakage of confidential information. This is especially relevant when the application is a monolithic one that accesses (via reading or writing) ordinary as well as confidential media. The technique partitions the application into two parts (each run as a separate program) with each part having its own confidentiality policy. The first program performs operations on public output channels, and the confidentiality policy does not allow it to read confidential information. The second program is allowed to read confidential information, but is not allowed to write to public channels. The authors contend that this partitioning enables them to enforce a confidentiality policy that in totality prevents leakage of confidential information from the original program on publicly observable channels.

## MetriCon 1.0

*Vancouver, B.C., Canada*

*August 1, 2006*

*Summary by Dan Geer*

[This material was excerpted from the digest found at www.securitymetrics.org.]

MetriCon 1.0 was held on August 1, 2006, as a single-day, limited-attendance workshop in conjunction with the USENIX Association's Security Symposium in Vancouver, British Columbia. The idea had been first discussed on the securitymetrics.org mailing list and subsequently an organizing committee was convened out of a lunch at the RSA show in February 2006. There was neither formal refereeing of papers nor proceedings, but there is both a digest of the meeting and a complete set of presentation materials available at the www.securitymetrics.org Web site. Andrew Jaquith (Yankee Group) was chair of the organizing committee, the members of which were Betsy Nichols (ClearPoint Metrics), Gunnar Peterson (Artec Group), Adam Shostack (Microsoft), Pete Lindstrom (Spire Security), and Dan Geer (Geer Risk Services).

### KEYNOTE ADDRESS

■ *Resolved: Metrics Are Nifty*
*Andrew Jaquith, Yankee Group*

■ *Resolved: Metrics Are Too Hard*
*Steve Bellovin, Columbia University*

Andrew Jaquith opened MetriCon 1.0 by pointing out that other fields have their bodies of managerial technique and control, but digital security does not, and that has to change. He presented his list of what features are included in a good metric: It must (1) be consistently measured, (2) be cheap to gather, (3) contain units of measure, (4) be expressed as a number, and (5) be contextually specific. Jaquith argued that this all breaks down to modelers versus measurers. Modelers think about how and why; measurers think about what. He was quick to admit that measurement without models will not ultimately be enough, but "let's get started measuring *something*, for Heaven's sake."

Steve Bellovin countered with the brittleness of software and thus the infeasibility of security metrics. Beginning with Lord Kelvin's dictum on how, without measurement, your knowledge is "of a meagre and unsatisfactory kind," Bellovin said that the reason we have not had much progress in measuring security is that it is in fact infeasible to measure anything in the world as we now have it. We cannot answer "How strong is it?" in the same style as a municipal building code unless we change how we do software. Because defense in the digital world requires perfection and the attacker's effort is linear in relation to the number of defensive layers, this brittleness will persist until we can write self-healing code. So his challenge: Show me the metrics that help this.

Lindstrom argued that Bellovin's reasoning did not show that metrics are impossible but rather that they are necessary. Another attendee asked, "So what if I agree on bugs being universal and it only takes one to fail a system? The issue is: How do we make decisions?" Butler agreed, saying that if it's that hopeless, then why do security at all? Epstein reminded all that formally evaluated systems still have bugs, too. Another attendee suggested that we can borrow some ideas from the physical world, especially relative measurements such as "this is safer than that." An attendee said that

there are certainly things you can measure, if for no other reason than to avoid stupid things. Another attendee cared about data, not software that handled data: "I want to know about changes in data state."

## SOFTWARE SECURITY METRICS

*Gunnar Peterson, track chair*

### *A Metric for Evaluating Static Analysis Tools*

*Brian Chess and Katrina Tsipenyuk, Fortify Software*

Peterson began with a call for rethinking what granularity we need if metrics are to be meaningful, not with regard to "system" or "security" but, rather, to C/I/A (confidentiality, integrity, and authentication).

Chess proposed a weighted composite score reflecting the orthogonal interests of the tool vendor, the auditor, and the developer and showed some preliminary results of applying this to a mix of tools and applications. He displayed real data from real work.

### *An Attack Surface Metric*

*Pratyusa Manadhata and Jeannette Wing, Carnegie Mellon University*

Manadhata posited a formal framework intended to find an answer to "Is the attack surface of A more serious than that of B?" Using the ratio of damage potential to attacker effort, he displayed several examples in each of which he manually annotated the source code and analyzed the call graph of the application using off-the-shelf tools. The question on the table is whether the number of vulnerabilities is or is not correlated with this attack surface metric.

### *"Good Enough" Metrics*

*Jeremy Epstein, WebMethods*

Rather than argue about which numbers it makes sense to collect, Epstein suggests gathering as many as you can and only then decide which make sense. Some numbers have only a distant relationship to vulnerabilities, some are merely retrospective, and some tend to too many false positives. Epstein suggested that ratios of Cowan's "relative vulnerability" sort are valuable, though Epstein's true desire is the security equivalent of "leading economic indicators."

### *Software Security Patterns and Risk*

*Thomas Heyman and Christophe Huygens, University of Leuven*

Huygens argued that we should attach metrics to security patterns, where a "pattern" is the observable connection between the core of one's computing environment and the ecosystem in which it lives. He is interested in ratio scores such as the number of firewall invocations vs. the number of service invocations, or the number of guards vs. the number of access points for each component. Preliminary results indicate that this approach is feasible; the aim is to craft indicators to use in the system design space.

### *Code Metrics*

*Pravir Chandra, Secure Software*

Chandra focused on remediation metrics—metrics that help (and assess) getting better and better. His main tool is a 4x4 matrix crossing severity (Critical, Error, Warning, Informational) with review state (Unknown, Known, Accepted, Mitigated). For each review state, he plans to use capture-recapture or capture-for-removal metrics to estimate flaw count and then look at changes in market share by severity to track progress. Chandra proposed correlating this with software complexity metrics (McCabe Cyclomatic, System, and Information Flow Complexity).

## ENTERPRISE AND CASE STUDIES A

*Adam Shostack, track chair*

Adam Shostack led off with a discussion of "Enterprise Case Studies: Substitute for Ongoing Data," followed by Butler on "What Are the Business Security Metrics?"

### *Data Breaches: Measurement Efforts and Issues*

*Chris Walsh*

Walsh began with the dates of (U.S.) adoption of data-breach laws and asked whether online breaches are a significant source of ID theft. Most studies focus on firm-level impact on income or stock price. To Walsh, such studies lack enough reach to establish causality or even to establish whether the public is simply becoming inured to breaches. More than anything else, Walsh was outlining what it is that we don't know and we will need to know, in other words, a research agenda.

### *The Human Side of Security Metrics*

*Dennis Opacki, Covestic*

To Opacki, the point of any metric is to change behavior, but behavior change is prone to pitfalls. Opacki tied together findings in evolutionary psychology, social psychology, and behavioral economics, noting that human intuition has low energy cost and runs in parallel, whereas human reason has high energy cost and runs single-threaded. Focus on scales that people gauge intuitively, keep the number of metrics small, do not neglect entertainment value, and give bad news first. Measure the impact of your delivery before and after, express everything in dollars where you can, and use plain language. Remember, it is better to be vaguely right than precisely wrong.

■ *No Substitute for Ongoing Data, Quantification, Visualization, and Story-Telling*

*John Quarterman and Gretchen Phillips, InternetPerils*

Quarterman demonstrated how his firm handles phishing attacks, making an argument for data aggregation. He used animation to illustrate a time series of complex interconnection paths. Quarterman is squarely in the observation (measurement) camp, not the simulation (modeling) camp, and he suggests collecting data when you do not yet need it so that when you do need it baselines are already in hand, such as on the other side of your firewall. Some discussion followed on what measured level of misbehavior an ISP needs before it can break its contract to support a phishing site.

■ *What Are the Business Security Metrics?*

*Shawn Butler, MSB Associates*

For Butler, business decisions are what security metrics are about. What we must have are frequency and impact if we are to get at true cost, and cost is the basis of business decision-making. Without impact, there is no importance to management. Irrationality is involved everywhere, driven, ironically, by standards of due care and the perception thereof. Butler does endorse the idea of decision support, but she notes that although requests are not coming down from on high, massive amounts of data are moving up and, worse, data represents lots of information about frequency (number of probes, viruses, unauthorized this or that) but nearly no information about impact (cost). She feels that impact is the hardest question to answer and that not assessing impact means there is no feedback between effectiveness and investment.

## ENTERPRISE AND CASE STUDIES B

*Betsy Nichols, track chair*

Nichols began the session by showing that maturity of security metrics deployment and market capitalization are uncorrelated. As session lead, she set out the core question: Why are metrics so hard? Her answer focused on three issues: vast and unclean data, a lack of consensus on indicators and models, and difficulty in packaging results.

■ *Leading Indicators in Information Security*

*John Nye, Symantec*

With his easy access to work at the Symantec Attack Center, Nye undertook to show what leading security indicators might look like. Beginning with the results of 449 remote penetration tests, he calculated a "vulnerability score" and "vulnerability saturation." Dividing his data set into quartiles by saturation, Nye showed an expectably sharp rise in vulnerability saturation from quartile to quartile. From that, he was able to identify specific vulnerabilities that might serve as leading indicators.

■ *Top Network Vulnerabilities Over Time*

*Vik Solem*

Solem used a similar data set limited to Nessus scans in a contiguous timespan, to identify the top 10 vulnerabilities over the study interval. Questions from the attendees mainly concerned details of data sources and methods. Using the Symantec Threat Report, Solem found no correlation between attacks and Nessus plug-in IDs, but there is correlation between attacks and what is in the Qualys "Laws of Vulns" report, although, as Opacki remarked, any tool including Nessus could be scanning for the wrong things.

■ *IAM Metrics Case Study*

*Andrew Sudbury, ClearPoint Metrics*

Sudbury confirmed that real work is hard; you must start with real goals and, within that, identify what is it that you do not know. His measures are designed to determine whether you are in control of your controls, and he confirmed that business value comes from fusing multiple data sources. Discussion was brisk: Kirkwood suggested that Sudbury add targets to his graphs of trend data. Jansen asked how one would confirm that a help desk clearance score is actually clearance and not just somebody skipping work. Blakley asked, Which of the following should be considered true? (1) Management is dumber than technical staff; (2) management and tech staff want to see different things; (3) you cannot give management bad news. Daguio said such tools let managers decide whether to ever give a particular team a project again.

■ *Assessment of IT Security in Networked Information Systems*

*Jonas Hallberg and Amund Hunstad, Swedish Defence Research Agency*

Hallberg made the insightful observation that, although system properties control the security level and the security level controls consequences, the security level is not measurable, whereas system properties and consequences are. Ergo, something that bridges the gap between system properties and likely or potential consequences has to be crafted. The Swedish Armed Forces uses five high-level security properties: access control, security logging, protection against intrusions, intrusion detection, and protection against malware. Saaty's "Analytic Hierarchy Process" was then used to differentially weight 20 low-level properties related to access con-

trol. The relationship between the properties was interesting and useful and was close to Bellovin's comments made at the beginning of the day.

## GOVERNANCE

*Dan Geer, track chair*

Geer set the tone for this session by simply declaring that the only metrics that matter are those for decision support in risk management.

### ■ *Model Concepts for Consideration and Discussion*

*Bryan Ware, Digital Sandbox*

Ware described how his firm calculates the U.S. Department of Homeland Security's allocation of grant dollars to municipalities. Before Ware's involvement, the criterion of per-capita dollars was used, which is fair but useless. But after Ware's involvement, risk-centric dollar allocation was used, which is easier said than done. The first step was to require management plans from states and cities that respond to the risk measures. Because 2x2 tables show the decisions you are making, Ware's firm used 17 sets of 6 experts each to work out criteria that set thresholds between high and low effectiveness (of proposed dollars spent) and high and low risk. Ware demonstrated how this was done, and subsequently how dollars were allocated—first to quadrants, then within individual quadrants. In the low/low quadrant, a minimum amount of money is used. In the high-risk but low-effectiveness quadrant, the two obvious East Coast U.S. cities were the most problematic. Choosing between low risk/high effectiveness and high risk/low effectiveness was hardest. Most money went to high effectiveness, which got Ware's firm raked over the coals. Discussion was brisk.

### ■ *Mission and Metrics from Different Views: Firm/Agency, Industry, and Profession*

*Kawika Daguio, Northeastern University*

Daguio reminded all to "Do no harm" as we introduce new metrics, that accountability matters, and that separating risk and compliance is essential. Compliance is more important than security's C/I/A requirements. A lot of what banks do is imposed on them, and the change from a compliance model to a risk model is a breath of fresh air. Daguio says that we should use nominal and ordinal measures to avoid bad effects and that we should not do interval or ratio scales because those invite comparison and hence organizational interference. Getting information sharing will require competitive, policy, technical, and political reasons for doing so, or it simply won't fly. Daguio was clear; although we are about metrics, these metrics do not exist in a vacuum nor are the recipients of the metrics necessarily going to be good-hearted and forthright. Discussion was again brisk.

### ■ *Measuring Information Security Risk*

*Bob Blakley, Burton Group*

Blakley began with a formal definition intended to disambiguate a measurement from a metric, and to look at metrics with an eye to finding "normal limits" and thus to act when you are outside them. In short, a measurement is something you take; a metric is something you give. He argues that we are not measuring risk, which is probability times impact. Instead of probability, we have to use game theory, and instead of measuring the probability of bad things, we have to measure consequence(s) of those bad things. Further, you use game theory to measure your opponent's goals as well as your own, which is a key point. Blakley illustrates this with a 2x3 matrix aimed at decision-making: high/low impact versus whether to mitigate, mitigate and recover, or recover alone. Blakley also pointed out that, for decision-making, correlates of risk are just as good as direct measures of risk, using as his example that while blood pressure, temperature, and pulse rate may not make you ill it is hard to make you ill without changing one or more of those three measures. He suggested we should find and be happy with such correlates in our sphere. There was then some discussion of frequentist versus Bayesian approaches and whether a bimodal probability distribution ($1 \times 10^6$ vs. $10^6 \times 1$) doesn't make any probabilist approach impossible. Quarterman asked about risk aggregation, and Butler reminded all that decision analysis is not about the "right" decision but about the "informed" decision, a meaningful difference. Geer and Blakley agreed that there is no probability distribution for a sentient opponent, so pure probability cannot be the answer.

### ■ *Information Assurance Metrics Taxonomy*

*Wayne Jansen, NIST*

Jansen showed one slide summarizing the taxonomy work of Vaughn et al. Jansen described himself as a novice in the metrics area and asked the audience to consider a number of questions drawn from the taxonomy. Does there exist somewhere a set of well-established metrics and measures on which a new organization should be focusing its initial efforts? The apparent discontinuity between strategic efforts and tactical ones leads one to ask whether there is a way to bridge the gap. What kinds of things need to be done to advance the state of the art? Do

we even know where we want to go? Ware answered that two of the most fascinating are the FICO (Fair Isaac) score, which made it possible to have instant credit decisions, and the KMV-Merton model to predict likelihood of default for corporations.

Daguio, as a banker at that time, asked that we all please not do something that wrenching again. He said that he had exhausted all the mechanisms he has for scoring security or something; the corporate end result is always to find a way to kill projects. An attendee asked whether it is a two-player game, or is game theory just intrinsically easier. Blakley answered that games are just as challenging and that what is going on now is, at least, a two-player game as illustrated by Microsoft's first Tuesday security drill and its monthly sequelae. Second, infosec is an economic game and not just a technical game. More discussion followed.

### DINNER/RUMP SESSION

Three unscheduled presentations rounded out the day: Leversage on "The Security Incident Database," Ozment and Schecter on "Does Software Security Improve with Age?" and Lindstrom on "Security Metrics."

Leversage observed that target of choice losses vastly exceed target of chance losses, that good old wiretapping is on the rise, that infected laptops as a transmission mechanism are very much on the rise, that human intelligence (HUMINT) is still the main source of information, and all in all his world is very much like the intelligence community. There is a growing demand from potential consumers, and it is private in every way.

Ozment described a fine-detail, time-series look at the history of OpenBSD. As this was a full conference paper with overlapping relevance to MetriCon, this summary is brief: Software does improve with age and is thus, as the title asks, more like wine than milk. As an inspired use of security metrics, this is a quotation from the paper's summary:

"We found statistically significant evidence that the rate of foundational vulnerability reports decreased during the study period. We utilized a reliability growth model to estimate that 67.6% of the vulnerabilities in the foundation version had been found. The model's estimate of the expected number of foundational vulnerabilities reported per day decreased from 0.051 at the start of the study to 0.024."

Lindstrom made a number of points about risk, using a number of Venn diagram examples of how to calculate varieties of risk. In Lindstrom's view, risk fluctuates the way a financial index like the S&P 500 fluctuates; as such, quantifying risk necessarily requires an actuarial tail (i.e., you calculate risk by looking at incidence and/or prevalence of activities in the past). That said, his examples are worth examining closely.

In summary, 44 people attended, predominantly representing industry (30) rather than academia (10) or government (4). Altogether the meeting lasted about 12 hours and ended on that note of happy exhaustion that marks a successful event. Not bad as a first try, and if you believe that imitation is the sincerest form of flattery, then MetriCon is already being flattered in more ways than one. If you want to be involved in this area, visit www.securitymetrics.org. Thanks go to USENIX for continuing its tradition of putting its trust in experiments.

## New Security Paradigms Workshop (NSPW '06)

*September 19–22, 2006*

*Schloss Dagstuhl, Germany*

The New Security Paradigms Workshop (NSPW) is a unique workshop devoted to the critical examination of new ideas in security. Each year since 1992, we have examined proposals for new principles upon which information security can be rebuilt from the ground up. Our program committee particularly looks for new paradigms: innovative approaches to older problems, early thinking on new topics, and controversial issues that might not make it into other conferences but deserve to have their try at shaking and breaking the mold.

The format of NSPW differs somewhat from other workshops. Attendance is limited to authors and workshop organizers, numbering around 30 total. All attendees are required to attend and pay attention to all presentations (no email, IM, or phone calls), without exception, so that all authors receive equal opportunity for discussion. We conduct extensive, highly interactive discussions of these proposals, from which we hope both the audience and the authors emerge with a better understanding of the strengths and weaknesses of what has been discussed. Free time outside of presentations is provided for those who have to conduct other business.

As opposed to most forums, where the authors present their papers and then answer a few questions afterward, NSPW allows questions to be asked during the presentation. As a result, although authors are given around 60 minutes for presentation, they are encour-

aged to limit their presentation material to 20 minutes and leave the rest of the time open for discussion. In some cases, authors presenting highly provocative or controversial topics may not make it past their second slide. We consider the high level of discussion to be the primary benefit of the conference, as stimulating discussion provides more feedback with which the author can refine his or her work.

Provocative work invites disagreement, especially work that is contrarian or questions the status quo. To prevent discussion from becoming an unfettered attack of the author's work, we engage the attendees in a "psychological contract," where positive feedback is strongly encouraged.

Since the discussion can provide significant feedback to the author, the final proceedings of the workshop are not published immediately at the workshop. Authors are given notes taken at their presentation, and they are expected to modify their papers based on the feedback they have received. The final proceedings are published two to three months after the workshop. The resulting papers are more complete and thought out than the original submissions.

Room and board is included in the registration fee, so that attendees can also share meals, easily participate in social activities, and not have to spend time traveling each day. This close interaction creates an atmosphere of camaraderie and provides for continued exchange of ideas.

It was my honor and pleasure this year to be the NSPW General Chair. I always find the workshop to be the most stimulating and highly enjoyable of all the workshops and conferences I attend. A terrific array of topics was presented this year, and a good time was had by all. In the following you will find a summary of the papers presented. I highly encourage researchers who have new paradigms to explore, especially risky or possibly "half-baked" ideas, to submit a paper to future New Security Paradigms Workshops.

*—Abe Singer, NSPW 2006 General Chair*

***Sessions summarized by Matt Bishop, Michael Collins, Carrie Gates, and Abe Singer***

### ■ *Hitting Spyware Where It Hurt$*

*Richard Ford and Sarah Gordon, Florida Institute of Technology*

The first paper outlined a method for retargeting click-fraud to damage spyware and adware vendors by increasing the risk associated with these methods. The authors develop a model for the return on investment for adware owners and then develop an attack aimed at disrupting the earnings of these owners by systematically sending fake requests.

The ensuing discussion focused on both the ethics and the logistics of implementing this network. An open question is the number of hosts that would be required to actually increase the risk to adware maintainers: A suggested biological analogy was the eradication of the Mexican Screw Worm in the 1970s, which was done by using sterile male Screw Worms who competed with the fertile male population. A huge number of infertile males was required to eradicate the fertile population, suggesting that an attack network would also have to be disproportionate.

### ■ *Dark Application Communities*

*Michael Locasto, Angelos Stavrou, and Angelos Keromytis, Columbia University*

This paper focused on the concept of a Dark Application Community (DAC), a botnet that forwards crash reports and other state disruptions to the bot maintainer. Essentially, the bot maintainer can acquire stack traces and other state disruptive information from normal use to acquire information on new potential vulnerabilities and threats that can then be used to generate exploitable code.

The ensuing discussion covered both the probability of successfully mining this technique for bugs and the implications for botnet management. It was pointed out that this technique extends a botnet's useful lifetime. An open question was whether this result would be more productive than fuzzing or other standard diversity techniques. Several noted the similarity between this method and n-version programming, although here the diversity is in usage rather than implementation. Possible experiments were suggested by comparing the bug discovery rates from open source auto-updated tools such as Firefox or Adium. A major concern was that, although generating reports was cheap, the cost of filtering and sorting the reports for valuable results was untenable.

### ■ *Challenging the Anomaly Detection Paradigm*

*Carrie Gates, CA Labs; Carol Taylor, University of Idaho*

This paper described weaknesses the authors perceived in the anomaly detection paradigm. The authors identified and questioned assumptions in three domains: the rarity and hostility of anomalies, problems in

training data, and incorrect assumptions about operational requirements.

In the first case, the authors argue that the assumptions made about the "normalcy" of data differ both since Denning's original studies and owing to changes in scope: Network data is more complex than system logs, and network data today is far more hostile than at the time of Denning's paper. In the second case, there are implicit assumptions about training data, such as the normalcy of a previous sample and the rarity of attacks that overlap this former case. Finally, the operational constraints were discussed in depth, with several commentators noting that the acceptable false-positive rate among the operational community is close to zero.

### *Inconsistency in Deception for Defense*

*Vicentiu Neagoe and Matt Bishop, UC Davis*

This paper questions whether deceptive mechanisms, such as servers and systems that present a false view of the system, need to maintain consistent views to fool attackers. It examined the nature of inconsistency in system response and actions. The deception model divides commands into two categories: do commands (which alter system state) and tell commands (which provide information on system state). Different implementations of deception were also presented.

Discussion focused on multilevel secure systems, where commands refuse to provide status information. How do these affect the model? If the probability of deception of each occurrence of events were independent, by repeating commands an attacker could probabilistically detect deception.

### *A Model of Data Sanitization*

*Rick Crawford, Matt Bishop, Bhume Bhuiratana, Lisa Clark, and Karl Levitt, UC Davis*

This paper introduced a competitive model of sanitization in the form of an inference game: a three-party game involving a sanitizer, an analyzer, and an adversary. The goal of sanitization (and the criteria for success in the game) is for the sanitizer to transform the data so that the analyst can obtain the desired information without the adversary obtaining any private information. An example was given using *k*-anonymity (e.g., a member of a set of *k* elements cannot be distinguished from any other member of the set).

Discussion focused on questions involving actively tampering with the dataset before releasing sanitized information. Examples of such attacks include salting the data beforehand and using the sanitization as a public excuse to announce something known privately (i.e., an insider requesting its own sanitized data).

### *Panel: Control vs. Patrol: A New Paradigm for Network Monitoring*

*Panelists: John McHugh, Dalhousie University; Fernando Carvalho-Rodrigues, NATO; David Townshed, University of New Brunswick*

The panelists debated the idea of an independent network-monitoring authority operating to ensure network integrity. The panelists contrast their concept of patrol versus more traditional discussions of network monitoring, which, in their perspective, are control- or ownership-oriented. The analogy driving the discussion was the role of highway patrols: Where a person drives in public spaces is their own business but that they were present is publicly accessible knowledge.

The ensuing discussion focused on two elements: the logistics of such a patrol mechanism and the role and implicit privacy of users. In the former case, there were fundamental questions of what the patrol would observe and collect. Some patrol functions already exist (e.g., chatroom chaperoning), but developing a large-scale patrol involves aggregating and analyzing huge volumes of data, and deciding what classes of problems the patrol would address. Given the initial concept of privacy on the highway, there was an extensive debate about the role of privacy online, with the recognition that a user's perception of privacy is extremely contextual and possibly totally unrelated to the facts on the ground (such as blogging using a public site).

### *Large-Scale Collection and Sanitization of Security Data*

*Phil Porras, SRI; Vitaly Shmatikov, UT Austin*

This paper summarized existing research challenges in data collection and sanitization for security research. Security research lacks a strong body of empirical work because of the lack of data sets; although public data sets are slowly being released, the question of sanitization is still not handled satisfactorily.

Discussion followed about how to handle the constraints of sanitization explicitly within the context of empirical research. Several suggestions focused around letting the sanitizer decide when data was released (e.g., whether some of these problems could be managed by releasing data sets after some safety period). As an alternative, a researcher may request logs of 3 consecutive days, but the sanitizer may decide which 3 consecutive days.

■ ***Googling Considered Harmful***

*Greg Conti, United States Military Academy*

The author began by showing AOLStalker, a search engine using the recently released (and then reclaimed) AOL dataset. This served as the context for the paper's thesis: Users increasingly rely on a large number of free services provided by a limited number of service providers. In the majority of cases, the price paid for these services is personal data: Users implicitly make micropayments of their personal privacy. The author developed a threat analysis model to privacy based on information released or gleaned from these services.

Discussion then followed on the various forms of signal analysis and social contracts previously used to protect privacy. Examples included tracking military mobilization by studying pizza deliveries in the D.C. area. Similarly noted were requirements to families of service members to keep silent before a deployment compared to the kind of logistic actions families may take en masse before a mobilization, such as communicating with various soldiers' benefits services. Discussion than focused on the construction of a privacy panel for W3C.

■ ***A Pact with the Devil***

*Mike Bond, University of Cambridge; George Danezis, KU Leuven*

The authors outlined a novel, and hypothetical, virus that would negotiate with its victim to improve its capacity to spread across networks. The hypothetical virus would offer an infected user a chance to commit a collaborative computer crime; for example, the original victim would write a mail that a new victim would readily open. In exchange for this, the virus would seek data on the new victim's drive (such as all of the new victim's email) and pass it on to the original victim.

Discussion focused on the strategies such a virus could take, and whether or not the victim could double-cross the virus. For example, in addition to offering carrots, the virus could eventually offer sticks such as threatening to release private or incriminating information, or planting criminal information on the victim's computer. Active comparisons were made to previous socially spreading problems (AIDS infections and the appearance of email chain letters on air-gapped networks being two prominent examples), along with consideration of what techniques would make the virus more effective, such as the scope of threats and offers the virus could make.

■ ***E-Prime for Security***

*Steve Greenwald, Independent Consultant*

This paper introduced E-Prime, a restricted subset of the English language developed by the General Semantics movement. E-Prime differs from English by avoiding all uses of the verb "to be," such as "is," "am," and "is not." The author argued that by eliminating these verbs, a writer is forced to provide more complete information, such as providing attribution to some action or requirement. Requiring that security policies be written in E-Prime would result in policies that are easier to read and that do not include assumed information. For example, "The administrator is required to provide audit logs" would become "The security team requires the administrator to provide audit logs."

■ ***Diffusion and Graph-Spectral Methods for Network Forensic Analysis***

*Wei Wang and Tom Daniels, Iowa State University*

This paper described a graph-theoretic approach to analyzing audit logs and network traffic with the aim of detecting attacks. The approach used was to have each node represent a host, for example, while connections between nodes would represent events. These events would have a weight associated with them that was based on some quality of the event or alert. The authors used eigenvectors to determine qualities of the network, finding that the first three eigenvectors often did not result in interesting information; however, the fourth eigenvector could isolate attacks.

The authors used data from the Lincoln Labs data set for testing, and so discussion focused on how this approach would perform given data from a real network. The primary issue discussed was what effect the noise inherent in a real network would have on the ability for this approach to extract attack information.

■ ***PKI Design for the Real World***

*Peter Gutmann, U. Auckland; Ben Laurie, Google; Bob Blakley, Burton Group; Mary-Ellen Zurko, IBM; Matt Bishop, UC Davis*

Each panelist described his or her belief about PKI and its adoption in the real world. Zurko started, describing the PKI system currently in use by Lotus Notes at IBM. This is a system that is deployed at many large enterprises and has been in use for several years. Laurie felt that the issue with PKI was the I: The infrastructure required for PKI was lacking. In particular, he noted that there were two requirements that needed to be met: (1) You want to know that

the person you are talking to today is the same person you were talking to yesterday, and (2) you want to know that the person you are talking to is the same person you were introduced to. Blakley, in contrast, felt that PKI was developed for two reasons: (1) Key distribution is hard and should be easier, and (2) digital signatures are cool. As a result, he felt that the main problem with PKI was that it was designed to solve a problem that no one had, and that PKI does not mimic any real-world processes. Bishop felt that the issue with PKI was that the design of the system is not understandable. For example, many nontechnical users do not understand what a chain of trust is. This also introduced legal issues, such as who has root and what does it mean to trust them? He felt that PKI was workable on a personal level (e.g., PGP) and at the level of a corporation, but not among the general public. Gutmann focused on a study he performed asking senior engineers and managers how they would design a PKI system given the constraint that they would need to support their design. He found that the systems described were all Web-based and differed completely from the designs proposed in the standards committee.

*Announcement and Call for Papers* **USENIX**

# 16th USENIX Security Symposium

***http://www.usenix.org/sec07***

**August 6–10, 2007** **Boston, Massachusetts**

## Important Dates

Paper submissions due: *February 1, 2007, 11:59 p.m. PST*
Panel proposals due: *March 29, 2007*
Notification to authors: *April 4, 2007*
Final papers due: *May 14, 2007*
Work-in-Progress reports due: *August 8, 2007, 6:00 p.m. EDT*

## Symposium Organizers

**Program Chair**
Niels Provos, *Google Inc.*

**Program Committee**
Kostas Anagnostakis, *Institute for Infocomm Research, Singapore*
Dan Boneh, *Stanford University*
Hao Chen, *University of California, Davis*
Monica Chew, *Google Inc.*
David Dagon, *Georgia Institute of Technology*
Marius Eriksen, *Google Inc.*
Kevin Fu, *University of Massachusetts Amherst*
Tal Garfinkel, *Stanford University*
Thorsten Holz, *University of Mannheim*
Somesh Jha, *University of Wisconsin*
Tadayoshi Kohno, *University of Washington*
Christopher Kruegel, *Technical University Vienna*
Wenke Lee, *Georgia Institute of Technology*
Patrick McDaniel, *Pennsylvania State University*
Fabian Monrose, *Johns Hopkins University*
Vern Paxson, *ICSI/LBNL*
Adrian Perrig, *Carnegie Mellon University*
Vassilis Prevelakis, *Drexel University*
Dug Song, *Arbor Networks*
Angelos Stavrou, *Columbia University*
Rebecca Wright, *Stevens Institute of Technology*
Paul Van Oorschot, *Carleton University*
Wietse Venema, *IBM Research*
Yi-Min Wang, *Microsoft Research, Redmond*

**Invited Talks Committee**
Bill Aiello, *University of British Columbia*
Angelos Keromytis, *Columbia University*
Gary McGraw, *Cigital*

## Symposium Overview

The USENIX Security Symposium brings together researchers, practitioners, system administrators, system programmers, and others interested in the latest advances in the security of computer systems and networks. The 16th USENIX Security Symposium will be held August 6–10, 2007, in Boston, Massachusetts.

All researchers are encouraged to submit papers covering novel and scientifically significant practical works in security or applied cryptography. Submissions are due on February 1, 2007, 11:59 p.m. PST. The Symposium will span five days: a two-day training program will be followed by a two and one-half day technical program, which will include refereed papers, invited talks, Work-in-Progress reports, panel discussions, and Birds-of-a-Feather sessions.

## Symposium Topics

Refereed paper submissions are solicited in all areas relating to systems and network security, including:

- Adaptive security and system management
- Analysis of network and security protocols
- Applications of cryptographic techniques
- Attacks against networks and machines
- Authentication and authorization of users, systems, and applications
- Automated tools for source code analysis
- Cryptographic implementation analysis and construction
- Denial-of-service attacks and countermeasures
- File and filesystem security
- Firewall technologies
- Forensics and diagnostics for security
- Intrusion and anomaly detection and prevention
- Malicious code analysis
- Network infrastructure security
- Operating system security
- Privacy-preserving (and compromising) systems
- Public key infrastructure
- Rights management and copyright protection
- Security architectures
- Security in heterogeneous and large-scale environments
- Security of agents and mobile code
- Security policy
- Self-protecting and healing systems
- Techniques for developing secure systems
- Technologies for trustworthy computing
- Voting systems analysis and security
- Wireless and pervasive/ubiquitous computing security
- World Wide Web security

Note that the USENIX Security Symposium is primarily a systems security conference. Papers whose contributions are primarily new cryptographic algorithms or protocols, cryptanalysis, electronic commerce primitives, etc., may not be appropriate for this conference.

Program committee members are limited to being authors or co-authors of at most two paper submissions. The program chair is not permitted to be author or co-author of any paper submissions.

## Refereed Papers & Awards

Papers that have been formally reviewed and accepted will be presented during the Symposium and published in the Symposium Proceedings. It is expected that one of the paper authors will attend the conference and present the work. It is the responsibility of the authors to find a suitable replacement presenter for their work, if the need arises.

One author per paper may take a registration discount of $200. If the registration fee poses a hardship to the presenter, USENIX can offer complimentary registration

#### Symposium Proceedings

The Proceedings will be distributed to attendees and, following the Symposium, will be available online to USENIX members and for purchase. The online Proceedings will be made available for free to everyone one year after the conference.

#### Best Paper Awards

Awards may be given at the conference for the best overall paper and for the best paper for which a student is the lead author. Papers by program committee members are not eligible for these awards.

## Training Program, Invited Talks, Panels, WiPs, and BoFs

In addition to the refereed papers and the keynote presentation, the technical program will include a training program, invited talks, panel discussions, a Work-in-Progress session (WiPs), and Birds-of-a-Feather sessions (BoFs). You are invited to make suggestions regarding topics or speakers in any of these sessions via email to the contacts listed below or to the program chair at sec07chair @usenix.org.

#### Training Program

Tutorials for both technical staff and managers will provide immediately useful, practical information on topics such as local and network security precautions, what cryptography can and cannot do, security mechanisms and policies, firewalls, and monitoring systems. If you are interested in proposing a tutorial or suggesting a topic, contact the USENIX Training Program Coordinator, Dan Klein, by email to tutorials@usenix.org.

#### Invited Talks

There will be several outstanding invited talks in parallel with the refereed papers. Please submit topic suggestions and talk proposals via email to sec07it@usenix.org.

#### Panel Discussions

The technical sessions may include topical panel discussions. Please send topic suggestions and proposals to sec07chair@usenix.org. The deadline for panel proposals is March 29, 2007.

#### Work-in-Progress Reports (WiPs)

One session of the Symposium will consist of Work-in-Progress reports (WiPs). This session offers short presentations about work in progress, new results, or timely topics. Speakers should submit a one- or two-paragraph abstract to sec07wips@usenix.org by 6:00 p.m. EDT on Wednesday, August 8, 2007. Make sure to include your name, affiliation, and the title of your talk. The schedule of presentations and accepted abstracts will be posted on the Symposium Web site. The time available will be distributed among the presenters, with each speaker allocated between 5 and 10 minutes. The time limit will be strictly enforced.

#### Birds-of-a-Feather Sessions (BoFs)

Birds-of-a-Feather sessions (BoFs) will be held Tuesday, Wednesday, and Thursday evenings. Birds-of-a-Feather sessions are informal gatherings of persons interested in a particular topic. BoFs often feature a presentation or a demonstration followed by discussion, announcements, and the sharing of strategies. BoFs can be scheduled on-site or in advance. To preschedule a BoF, please send email to the USENIX Conference Department at bofs@usenix.org with the title and a brief description of the BoF; the name, title, affiliation, and email address of the facilitator; and your preference of date and time.

## How and Where to Submit Refereed Papers

Papers are due by February 1, 2007, 11:59 p.m. PST. All submissions will be made online, and details of the submissions process will be made available on the Call for Papers Web site, http://www .usenix.org/events/sec07/cfp, well in advance of the deadline. Submissions should be finished, complete papers.

*Paper submissions **must not** be anonymized.*

Submissions must be in PDF format (i.e., processed by Adobe's Acrobat Distiller or equivalent). Note that LaTeX users can use the "dvipdf" command to convert a DVI file into PDF format. Please make sure your submission can be opened using Adobe Acrobat 4.0. For more details on the submission process, authors are encouraged to consult the detailed author guidelines.

To insure that we can read your PDF file, authors are urged to follow the NSF "Fastlane" guidelines for document preparation, and to pay special attention to unusual fonts. For more details, see:

- https://www.fastlane.nsf.gov/documents/pdf_create /pdfcreate_01.jsp
- https://www.fastlane.nsf.gov/documents/tex/tex_01.jsp

All submissions will be judged on originality, relevance, correctness, and clarity. In addition to citing relevant published work, authors should relate their submission to any other relevant submissions of theirs in other venues that are under review at the same time as their submission to the Symposium. Simultaneous submission of the same work to multiple venues, submission of previously published work, and plagiarism constitute dishonesty or fraud. USENIX, like other scientific and technical conferences and journals, prohibits these practices and may, on the recommendation of a program chair, take action against authors who have committed them. In some cases, program committees may share information about submitted papers with other conference chairs and journal editors to ensure the integrity of papers under consideration. If a violation of these principles is found, sanctions may include, but are not limited to, barring the authors from submitting to or participating in USENIX conferences for a set period, contacting the authors' institutions, and publicizing the details of the case.

Authors uncertain whether their submission meets USENIX's guidelines should contact the program chair, sec07chair@usenix .org, or the USENIX office, submissionspolicy@usenix.org.

Papers accompanied by nondisclosure agreement forms will not be considered. All submissions are treated as confidential, both as a matter of policy and in accord with the U.S. Copyright Act of 1976.

Authors will be notified of acceptance by April 4, 2007. The camera-ready final paper due date is May 14, 2007. Each accepted submission may be assigned a member of the program committee to act as its shepherd through the preparation of the final paper. The assigned member will act as a conduit for feedback from the committee to the authors.

Specific questions about submissions may be sent via email to the program chair at sec07chair@usenix.org.

## Program and Registration Information

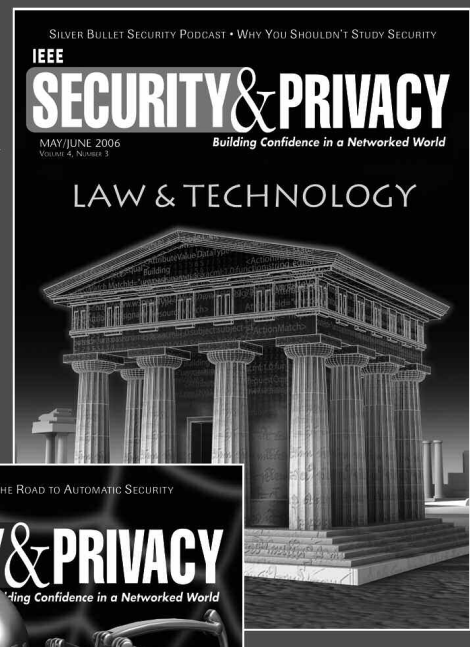
Complete program and registration information will be available in May 2007 on the Symposium Web site, both as HTML and as a printable PDF file. If you would like to receive the latest USENIX conference information, please join our mailing list at http://www .usenix.org/about/mailing.html.

Last Updated: 9/26/06

# New nonmember rate of $29 for *S&P* magazine!

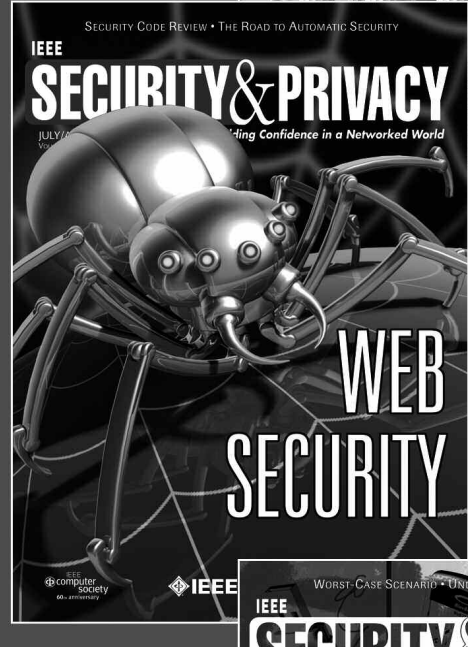
*IEEE Security & Privacy* magazine is the premier magazine for security professionals. Each issue is packed with information about cybercrime, security & policy, privacy and legal issues, and intellectual property protection.

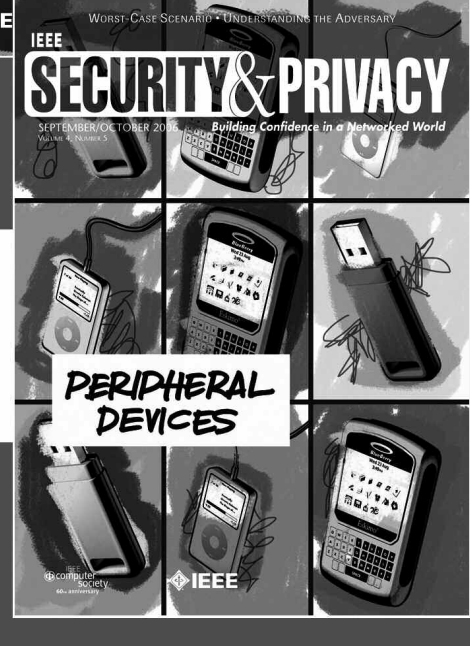
*S&P* features regular contributions by noted security experts, including Bruce Schneier & Gary McGraw.




Top security professionals in the field share information you can rely on:

- Wireless Security
- Intellectual Property Protection and Piracy
- Designing for Infrastructure Security
- Privacy Issues
- Legal Issues
- Cybercrime
- Digital Rights Management
- Securing the Enterprise
- The Security Profession
- Education




## Save 59% off last year's price!

www.computer.org/services/nonmem/spbnr

;login:

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