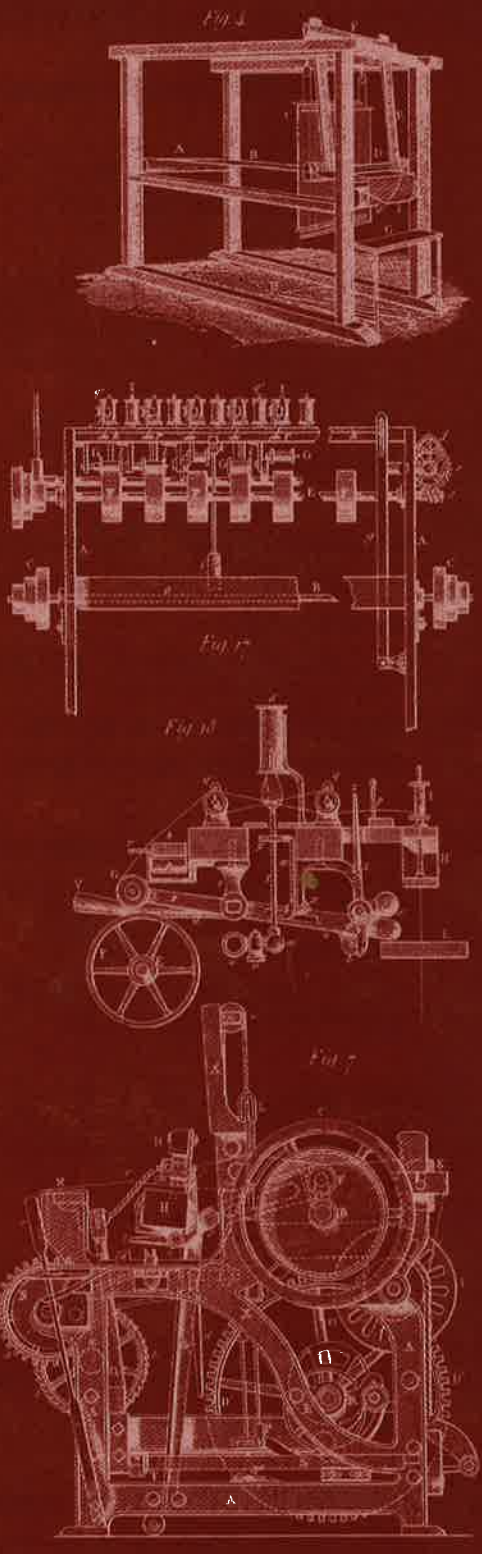
APRIL 2005 VOLUME 30 NUMBER 2

;login:

THE USENIX MAGAZINE



USENIX
The Advanced Computing Systems Association

USENIX Upcoming Events

2005 USENIX Annual Technical Conference

APRIL 10–15, 2005, Anaheim, CA, USA
http://www.usenix.org/usenix05

2nd Symposium on Networked Systems Design and Implementation (NSDI '05)

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

MAY 2–4, 2005, Boston, MA, USA
http://www.usenix.org/nsdi05

3rd International Conference on Mobile Systems, Applications, and Services (MobiSys '05)

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

JUNE 6–8, 2005, Seattle, WA, USA
http://www.usenix.org/mobisys05

First ACM/USENIX International Conference on Virtual Execution Environments (VEE '05)

Sponsored by ACM SIGPLAN and USENIX, in cooperation with ACM SIGOPS

JUNE 11–12, 2005, Chicago, IL, USA
http://www.veeconference.org

10th Workshop on Hot Topics in Operating Systems (HotOS X)

JUNE 12–15, 2005, Santa Fe, NM, USA
http://www.usenix.org/hotos05

Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI '05)

JULY 7–8, 2005, Cambridge, MA, USA
http://www.usenix.org/sruti05

Linux Kernel Developers Summit 2005

JULY 17–19, 2005, Ottawa, Ontario, Canada
http://www.usenix.org/kernel05

14th USENIX Security Symposium (Security '05)

AUGUST 1–5, Baltimore, MD, USA
http://www.usenix.org/sec05

Internet Measurement Conference 2005 (IMC '05)

Sponsored by ACM SIGCOMM, in cooperation with USENIX

OCTOBER 19–21, 2005, New Orleans, LA, USA
http://www.usenix.org/imc05
Abstracts due: May 6, 2005

ACM/IFIP/USENIX 6th International Middleware Conference

NOVEMBER 28–DECEMBER 2, 2005, Grenoble, France
http://middleware05.objectweb.org

19th Large Installation System Administration Conference (LISA '05)

Sponsored by USENIX and SAGE

DECEMBER 4–9, 2005, San Diego, CA, USA
http://www.usenix.org/lisa05
Extended abstracts due: May 10, 2005

4th USENIX Conference on File and Storage Technologies (FAST '05)

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

DECEMBER 14–16, 2005, San Francisco, CA, USA
http://www.usenix.org/fast05
Paper submissions due: July 13, 2005

2006 USENIX Annual Technical Conference

MARCH 12–17, 2006, Boston, MA, USA

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

contents

;login:

VOL. 30, #2, APRIL 2005

EDITOR
Rob Kolstad
rob@usenix.org

CONTRIBUTING EDITOR
Tina Darmohray
tmd@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
Steve Gilmartin
proofshop@usenix.org

PRODUCTION STAFF
Casey Henderson
Alex Walker

TYPESETTER
Star Type
startype@comcast.net

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

office@usenix.org
login@usenix.org
conference@usenix.org
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.

ROB KOLSTAD

motd

LEVERAGE: GETTING RESULTS

Dr. Rob Kolstad has long served as editor of *;login:*. He is SAGE's Executive Director, and also head coach of the USENIX-sponsored USA Computing Olympiad.

- *kolstad@usenix.org*

Given our finite working life, the concept of leverage holds interest for those who wish to accomplish more things, to "get more done." In our careers as computer professionals, we see leverage and its application (or misapplication) continually.

The dictionary says that "leverage" is a synonym for both "effectiveness" and "power," and that feels about right to me. The whole computer industry revolves around paying a (presumably) low cost for an engine or machine that will amplify effectiveness for a person or group and supplementing the hardware with software that customizes that machine for a particular environment.

The airline reservations industry is a fabulous example of this approach. Imagine 10,000 people shuffling index cards that represent the availability of seats on the myriad airplanes flying every day for, say, the next year. That image is almost laughable. Nowadays, customers choose seating preferences and request special meals, in addition to seeing several dozen flight possibilities for their journey, with a minimal number of clicks. The power of databases coupled with the Internet and home computing engines enables this solution as the quickest, most accurate, and presumably cheapest. This is a great example of leveraging a computer and software to attack a problem.

Software is the medium by which computer leverage is conveyed. Software's creators intend to endow their users with specific skills or power. Their software tends to provide not only specific attacks (algorithms) on problems but also paradigms that offer useful approaches. The choice of a suitable computing environment can confer astounding leverage on folks who otherwise could never dream of attempting a complex solution.

Another kind of leverage is people leverage, commonly called management. Making the transition from enjoying personal accomplishments to enjoying the total set of accomplishments is a difficult one (or at least it was for me). "You didn't do anything but sit in your office," say the skeptics. Planning, encouragement, hiring, provisioning, and all the other myriad tasks are ignored by those who take them for granted. Nevertheless, it's easy to see that some people might feel pride of accomplishment in projects they champion, manage, and somehow move to completion. This is serious leverage.

Management leverage appears in a variety of situations and locales, ranging from large institutions like General Motors or Harvard University to smaller organizations like a church or a model railroading

club. Each of these milieus offers a different kind of potential leverage and concomitant reward.

Another kind of leverage is seen more on the political front and is potentially exemplified in one way by the open source movement. In *increasing* order of degree of leverage, the open source promoters provide:

- software
- communication
- documentation and training
- advocacy

Why is advocacy the highest-leverage activity? Because it is the one that wins the hearts and minds of those who only then will use the other three to attack whatever problem is important to them.

This is not to say, of course, that advocacy for its own sake has a lot of value. Open-mindedness often trumps tunnel vision when trying to solve problems.

What difference does all this make? It makes a difference if your personal motivators include concepts such as effectiveness, solution completion, and getting things done.

Thinking in terms of leverage means asking questions such as: "Is this a good way to spend my time?" "Will this action be as effective as [some other action]?" "If [Alice and Bob] do this, can I attack some issue or problem that they are unable or unwilling to attack?" Obviously, it also revolves around answering those questions with some level of usefulness.

I believe those who enjoy accomplishment may also enjoy leverage. It can lead to a tremendous amplification of personal skills and, best of all, make the world a better place. After all, that's the goal, isn't it?

USENIX Membership Update

Membership renewal information, notices, and receipts are now being sent to you electronically!

Remember to print your electronic receipt, if you need one, when you receive the confirmation email.

If you have not provided us with an email address, you are welcome to update your record online.

See http://www.usenix.org/membership.

You are welcome to print your membership card online as well. The online cards have a new design with updated logos—all you have to do is print!

Please note that some annual membership dues increased as of February 1, 2005:

- Individual membership dues: $115.
- Educational membership dues: $250.
- Corporate membership dues: $460.
- Student and Supporting memberships remained at 2004 rates.
- SAGE membership dues remained the same at $40.

letter to the editor

TO RIK FARROW:

Rik,

Very likely someone has already mentioned sfs—http://www.fs.net/sfswww/sfsfaq.html—to you. But just in case . . .

Cheers,

MARKO
schutz_m@usp.ac.fj

RIK FARROW REPLIES:

I am aware of SFS, as Kevin Fu wrote an article about it for the Security edition of *;login:* (which I edited) four or five years ago.

SFS does provide strong security over untrusted networks. It also works mainly for anonymous access, as its strong point is self-certification of the server, not of the user accessing the server. Users can authenticate using sfsagent, then log out and back in again.

For most organizations that are using NFS, SFS would work well enough, but the extra hoops needed for user authentication (and if there are multiple SFS servers, on each server) are an obstacle.

My column was about old, generally ignored failings to authenticate users in NFS, which many people are still using. It was the incident at SDSC that got me interested in this again. I want people to examine what they are currently using, rather than championing a new approach. There are quite a few people working on improving NFS, whereas SFS is still considered alpha software, and the last update was April 23, 2003.

I hope we have succeeded in reminding people of the existence of SFS, so they can make their own decision.

RIK FARROW
rik@spirit.com

STEVEN ALEXANDER

finding malware on compromised Windows machines

Steven is a programmer for Merced College. He manages the college's intrusion detection system.

■ *alexander.steven@sbcglobal.net*

This article discusses possible responses to suspicious activity on Windows machines. It surveys several free tools that are useful in documenting the current state of a system and in detecting and analyzing suspicious processes or open network ports.

Sometimes it is obvious that a machine has been compromised (e.g., when a Web site is defaced). Other times, suspicious behavior is detected that warrants further investigation. This activity could be the result of a break-in, a virus or worm, spyware, or something more benign. If the suspicious behavior is the result of a break-in, law enforcement may need to be contacted (depending on your organization's policy). For this reason, it is important that volatile information be saved and that every step you take is documented. The tools discussed in this article can be used to document the current state of a system that has been compromised or to investigate a system that is behaving suspiciously.

I was recently called upon to investigate two Windows machines that had been exhibiting suspicious behavior. The first thing I did was to create a CD with several tools I thought would be useful in analyzing the system and documenting the state of the system. As it turns out, the machines in question had not been compromised by a human intruder but by two different worms; these two systems were not properly patched.

Starting Out

The first thing I do when investigating any system (Windows, UNIX, or otherwise) is glean lists of the users who are logged on, running processes, and network connections. System administrators should use these tools on freshly installed systems and on production systems to determine what a normal system looks like. If you don't know what should be on your system, it is very difficult to figure out what shouldn't be on your system.

After gathering basic information, I check the Windows registry and the Windows startup folders to see what is starting up with the system. Sometimes it is easy to determine what each of the programs that are scheduled to start automatically actually is. Unfortunately, a lot of legitimate software vendors like to do asinine things such as stick an executable with a weird name into C:\windows or C:\windows\system32. This can make it difficult to determine whether a program is legitimate. The best thing to do is to Google for the filename and see what turns up.

The Reg tool, described below, can be used to dump the appropriate keys from the registry. Any undesired entries can be removed using Regedit or, on Windows XP, Msconfig. I strongly suggest using Msconfig on XP systems, since it enables you to uncheck an entry but, if you wish, restore the entry later. If you are using Regedit, back up the registry before deleting anything. Also, rather than deleting any programs referenced by these entries, move and/or rename them to avoid losing something you need. Administrators should try to become familiar with the software that should be starting automatically on their servers and workstations. Again, it's hard to determine what is anomalous if you don't know what normal is.

My approach when investigating a system is to look for anything that does not belong. This includes spyware, worms, back doors, etc. A centrally managed antivirus program is a good way to detect a lot of malware as soon as it enters the system, but it won't detect everything. Netcat, for instance, is not malware and won't be detected by an antivirus program, but it can be used to bind cmd.exe to a port for use as a back door.

This article is limited to discussing the tools and procedures that can help determine whether a system has been compromised. Responding to a compromise is an even larger issue (one I hope to cover in another article). Still, a response policy—vetted by upper management—must be in place before an incident occurs. Some important points that must be decided include who is responsible for the technical response, how evidence will be handled, who decides whether to contact law enforcement, and whether it matters if the intrusion occurred from within or from outside your organization.

It is also helpful to find out from your local law enforcement agency what they want you to do in the event of a break-in. At what point do they want to be contacted? How should you proceed? How should you preserve possible evidence? Whom, in law enforcement, should you contact? If you simply call your local police department after a break-in, the response will probably be from a uniformed officer who has no training in these matters. Often, you will want to contact the detective or group that handles computer evidence or computer crimes.

See references [1–4] for useful reading on incident response and forensics.

In the following sections, I give brief descriptions of several tools, all freely available, that I've found useful. With the exception of Microsoft's Reg tool, they are available from SysInternals [5].

The Tools

WINDOWS: REG

Reg is a Windows utility that can be used to extract data from the Windows registry. A large number of malware programs add an entry in the registry so that they will be started automatically if the computer reboots. Under most circumstances (if you're not currently installing something), the RunOnce and RunOnceEx keys should be empty. Track down any programs listed under the Run keys and make sure they represent legitimate software.

```
reg query "HKEY_CURRENT_USER\Software\Microsoft\Windows\Currentversion\Run" /s
reg query "HK_LOCAL_MACHINE\Software\Microsoft\Windows\Currentversion\RunOnceEx" /s
reg query "HK_LOCAL_MACHINE\Software\Microsoft\Windows\Currentversion\RunOnce" /s
reg query "HK_LOCAL_MACHINE\Software\Microsoft\Windows\Currentversion\Run" /s
```

SYSINTERNALS: PSLIST

PsList is similar in function to the UNIX ps command. It displays a list of the running processes on the system, including the process ID, priority, and number of threads. With the -m option, PsList also displays memory-usage information, including the working set size. With the -t option, the running processes are displayed in a process tree instead of a flat list.

```
PsList 1.26 - Process Information Lister
Copyright (C) 1999-2004 Mark Russinovich
Sysinternals - www.sysinternals.com

Process information for C37163ALEXANDER:
```

Name	Pid	Pri	Thc	Hnd	Priv	CPU Time	Elapsed Time
Idle	0	0	1	0	0	85:28:28.343	0:00:00.000
System	4	8	80	292	0	0:04:00.593	0:00:00.000
smss	452	11	3	19	164	0:00:00.046	143:29:16.906
csrss	524	13	11	407	1648	0:02:12.312	143:29:12.890
winlogon	548	13	19	581	7728	0:00:09.875	143:29:11.250
services	592	9	16	286	4244	0:00:15.750	143:29:09.187
lsass	604	9	17	377	2468	0:00:06.515	143:29:09.046
svchost	796	8	5	133	1360	0:00:00.109	143:29:06.093

SYSINTERNALS: HANDLE

Handle is a command-line utility that shows the handles open by every process on the system. When used without options, Handle only displays open file handles. When used with the -a option, Handle displays open handles to all objects, including files, registry keys, processes, ports, and semaphores. I suggest dumping all open handles to one file and dumping just open file handles to a second file for convenience. Handle is very useful for figuring out what a program is actually doing.

```
Handle v2.2
Copyright (C) 1997-2004 Mark Russinovich
Sysinternals - www.sysinternals.com
------------------------------------------------------------
smss.exe pid: 452 NT AUTHORITY\SYSTEM
     8: File         C:\WINDOWS
    1c: File         c:\WINDOWS\system32
------------------------------------------------------------
csrss.exe pid  524 NT AUTHORITY\SYSTEM
     c: File         C:\WINDOWS\system32
    38: section        \NLS\NlsSectionUnicode
    40: Section        \NLS\NlsSectionLocale
    44: Section        \NLS\NlsSectionCType
    48: Section        \NLS\NlsSectionSortkey
    4c: Section        \NLS\NlsSectionSortTbls
   2e0: Section        \BaseNameObjects\ShimSharedMemory
   564: File         c:\WINDOWS\system32\ega.cpi
```

SYSINTERNALS: SYSLISTDLLS

ListDLLs is a command-line utility that displays the DLL files loaded by each process running on the system. The utility shows the full path of each DLL. I find that the path information is particularly helpful because it helps me to identify what application or service a process is associated with.

```
ListDLLs v2.25 - DLL lister for Win9x/NT
Copyright (c) 1997-2004 Mark Russinovich
Sysinternals - www.sysinternals.com
------------------------------------------------------------
```

```
System pic: 4
Command line: <no command line>
------------------------------------------------------------------------------

smss.exe pid: 452
Command line: \SystemRoot\System32\smss.exe

Base        Size      Version         Path
0x48580000  0xf000                    \SystemRoot\System32\smss.exe
0x7c900000  0xb0000   5.01.2600.2180  C:\WINDOWS\system32\ntdll.dll
------------------------------------------------------------------------------

winlogon.exe pid: A548
Command line: winlogon.exe

Base        Size      Version         Path
0x01000000  0x80000                   \??\C:\WINDOWS\system32\winlogon.exe
0x7c900000  0xb0000   5.01.2600.2180   c:\WINDOWS\system32\ntdll.dll
0x7c800000  0xf4000   5.01.2600.2180   c:\WINDOWS\system32\kernel32.dll
0x77dd0000  0x9b000   5.01.2600.2180  c:\WINDOWS\system32\ADVAPI32.dll
0x77e70000  0x91000   5.01.2600.2180   c:\WINDOWS\system32\RPCRT4.dll
0x776c0000  0x11000   5.01.2600.2180   c:\WINDOWS\system32\AUTHZ.dll
0x77c10000  0x58000   7.00.2600.2180   c:\WINDOWS\system32\msvcrt.dll
```

SYSINTERNALS: TCPVCON

Tcpvcon displays a list of all established TCP connections along with their owning process. When used with the -a option, it will display all connection endpoints (TCP and UDP), established or not.

```
[TCP] C:\Program Files\Netscape\Netscape\Netscp.exe
    PID        2676
    State:     ESTABLISHED
    Local:     c37163alexanders:3283
    Remote:    c37163alexanders:3284
[TCP] C:\Program Files\Netscape\Netscape\Netscp.exe
    PID:       2676
    State:     ESTABLISHED
    Local:     c37163alexanders:3284
    Remote:    c37163alexanders:3283
```

PSLOGLIST

Psloglist displays the contents of the event logs. By default, Psloglist dumps the system log, but it can be used to dump the other logs by running psloglist logname. If used to dump the Directory Service or File Replication Service logs, quote the name on the command line. The program is also capable of dumping records from after a specified date by running it with the -a option, e.g., psloglist security -a 01/01/05.

It is essential to remember that your system needs to be configured to log important events in the first place. The standard events that are logged by Windows simply do not provide you with enough detail about a break-in.

At a minimum, enable auditing for policy change, privilege use, and logon events in the Local Security Policy under Administrative Tools in the Control Panel (this can also be accessed using mmc). You may also wish to audit access to important or confidential data (this can be configured by right-clicking on any file or folder, choosing Properties, and clicking Advanced under the Security tab).

```
[005] Security
    Type:      AUDIT SUCCESS
    Computer:  SEGFAULT
    Time:      12/30/2003 3:34:52 PM   ID:      643
    User:      MCCEDU\alexander.s
```

```
Domain Policy Changed: Password Policy modified
    Domain Name:        SEGFAULT
    Domain ID:          %{S-1-5-21-123456789-123456789-123456789}
    Caller User Name:   alexander.s
    Caller Domain:      MCCEDU
    Caller Logon ID:    (0x0,0xC6EF)
    Privileges:         -
```

Conclusion

Your best tool is wetware. The more familiar you are with the normal processes and services running on your systems, the easier it will be to detect anything out of place. Also, the Event Logs are of little use unless you enable additional auditing.

Some intrusions are hard to detect. If your firewall logs or IDS indicates that there may be a problem with a machine and your initial investigation turns up nothing, you may wish to crank up the logging for a spell and see what turns up.

I do not recommend blindly searching the file system unless you are willing to image the drives on the system (or, less preferably, make a tape backup) before searching, since you might inadvertently modify the file access times. I know of one utility that is supposed to be able to search for files accessed within a given time range without updating the attributes, but that utility fails to find many files.

Not searching the file system has the drawback that you may miss the signs of a break-in. Nevertheless, I would rather increase logging and wait things out. If the attacker is discreet, you may miss whatever he has left behind anyway.

Sometimes it is known that a system has been compromised, but it is not known whether the attack is the work of self-propagating malware or a human intruder. If you don't find any evidence of a virus or worm that accounts for previously observed events, it may be appropriate to treat it as a human attack until proven otherwise. At this point, you should take the affected system offline and image the drives. If you do find malware on the system, make sure that particular malware accounts for any observed events and is not independent of or a cover for a human attack.

POSTSCRIPT

Since this article was written, SysInternals has released a new tool, Rootkit Revealer. It looks promising: check it out.

REFERENCES

[1] Jamie Morris, "Forensics on the Windows Platform, Part One," http://www.securityfocus.com/infocus/1661.

[2] Jamie Morris, "Forensics on the Windows Platform, Part Two," http://www.securityfocus.com/infocus/1665.

[3] H. Carvey, "Win2K First Responder's Guide," http://www.securityfocus.com/infocus/1624.

[4] Chris Prosise, Kevin Mandia, and Matt Pepe, "Incident Response and Computer Forensics," McGraw-Hill Osborne Media, 2003.

[5] Mark Russinovich and Bryce Cogswell, SysInternals, http://www.sysinternals.com.

RIK FARROW

musings

Rik Farrow provides UNIX and Internet security consulting and training. He is the author of *UNIX System Security* and *System Administrator's Guide to System V,* and editor of the SAGE Short Topics in System Administration series.

■ *rik@spirit.com*

Corruption. The very thought sends shivers up and down my spine. And that is the goal of those who would break into your systems, so they can "own" them. They want to take control of your systems, preferably in a manner that is difficult to detect. Out of this desire came rootkits: corruption made simple.

I got my first rootkit from a friend at a university, my source for lots of examples of stuff left behind on compromised systems (nice, delicate term for being hacked). That rootkit was one of the first written, and contained trojans for SunOS 4. In the README file, the author of the rootkit had written (approximately), "I got tired of doing the same things over and over again, so I packaged them up." The rootkit contained trojans designed to hide the presence of certain files, processes, network connections, and a network sniffer. If you remember what networks and network protocols were like in 1993, you'll understand why this sniffer worked very well at collecting usernames and passwords.

Over the years, people added features to rootkits, such as the ability to edit logfiles or, better yet, prevent certain log entries from being appended to logfiles by trojaning the syslog daemon. New commands were added to the list of trojans. But the worst was yet to come.

The problem with command-level trojans is that it is relatively easy to detect them. Tools like Tripwire were written specifically with this in mind, as installation of trojans and other malware became commonplace. Most trojans rely on access to source code, and that leads to trojans for closed source systems being based upon open source software. If someone used the BSD source to ls, for example, the flags and behavior would not be the same as they would be for AIX or HP/UX. Close, but not exact. And systems like Solaris don't have just one version of ls, but several.

Going Deep

The solution, from the perspective of an attacker, was to move the rootkit deeper. If the rootkit runs at the kernel level, then nothing can be trusted. All software, whether on UNIX, Windows, Linux, or *BSD, relies on the kernel for all access to resources such as files, sockets, memory, and new processes. The system call interface provides this access. In UNIX-like systems, the system call interface provides a couple of hundred entry points for doing things like listing directories, files, programs, sockets, and active processes (189 in

OpenBSD 3.4, 315 in Linux 2.6). In Windows NT and its descendants, the number of entry points is more than 2000, but the concept is the same. In either case, if the attacker can insert code into the kernel, that attacker has the deepest level of control over a system.

The obvious way to insert code is to modify the kernel source directly. But there is a problem with that approach, in that a system must be rebooted before the changes take effect, and rebooting a UNIX-like system is rare enough that it would be noticed (in most cases). But there is also an obvious solution—use a method that permits patching the operating system without rebooting.

You have certainly heard of loadable kernel module (LKM) rootkits. LKMs permit sysadmins to install software in an operating system without rebooting it, or to configure a kernel at boot time without having all possible devices already linked into the kernel. While LKMs are convenient for sysadmins, they are just as convenient for any attacker who has acquired root access and wants to install the best in rootkit technology.

And Deeper

Over time, even LKM rootkit technology has improved. Early versions worked by replacing function addresses in the system call table with their own entry points. The original system call function still gets called, but the results of the system call get filtered to hide whatever the rootkit designer wants to hide. Initially, this was pretty much the same stuff that was done in the original, SunOS, command-level rootkit. But then it started to change.

One creative use of kernel-level rootkitting was file redirection. If you ran an integrity-checking tool like Tripwire (or anything that read a file), you would get the original version of the file. But if that file contains a program, when a request was made to execute it, a different program got run instead.

LKM rootkits can perform privilege elevation. In many of the rootkits around today (e.g., adore, adore-ng, all-root, kbdv3, rkit, shtroj2, and synapsys), the rootkit installer can either get a root shell or run a program as root by using whatever key the rootkit requires. In adore-ng, echoing the adore key to /proc elevates the privilege and capabilities of the shell to root without restrictions. This beats the pants off the old, SunOS rootkit technique of using back doors in SUID files like chsh and passwd. Adore-ng also prevents log records of hidden processes from being written.

Even the methods used to hide things have changed. Adore-ng, instead of hooking system calls, actually hooks into the Virtual File System (VFS) interface to perform its deeds. This works because both files and processes get listed via the VFS in Linux and some other operating systems (adore-ng works only on Linux). You can read Phrack (http://www.phrack.org/phrack/58/p58-0x06) if you want to learn how this is done.

Adore-ng also offers a new technique for hiding its own presence. The adore-ng.o file can be linked with an existing kernel module, so that when that module gets loaded at boot time, so will adore-ng. This makes adore-ng much more difficult to detect, and quite neatly solves the problem for the attacker of how to reload it after the next reboot. For details, you can check out Phrack again (http://www.phrack.org/phrack/61/p61-0x0a_Infecting_Loadable_Kernel_Modules.txt). It turns out neither to be difficult nor difficult to understand, and relies on a documented feature of ld plus a little symbol name manipulation.

By moving the hooks into a deeper level of the file system, tools that monitor the system call table for changes will miss the installation of rootkits like adore-ng. I did uncover a paper by Kruegel, Robertson, and Vigna (http://www.cs.ucsb.edu/

~vigna/pub/2004_kruegel_robertson_vigna_ACSAC04.pdf) that performs binary analysis of LKMs and detects rootkits by checking for the memory they seek to modify. Most LKMs stick to the regions of memory that a device driver would need to modify in order for initialization to succeed, but not rootkits, which stray to regions only miscreants would go. Certainly an interesting approach.

Another "interesting approach" comes in the form of SUCKIT, a kernel-level rootkit that does not rely on using LKM hooks. This charmingly named rootkit does its work by reading and writing directly to /dev/kmem. Unlike the LKM approach, which relies on being able to locate the kernel symbol tables, this rootkit searches through kernel memory looking for the pattern of bytes typically found within the soft interrupt handler, the entry point to the kernel and the system call table. The soft interrupt handler address can be gleaned from a single Intel assembler instruction, sidt %0, and then the code searches for the offset to the actual call to the system call table. You can read about this in Phrack too: http://www.phrack.org/show.php?p=58&a=7.

So, even if you compile a kernel without LKM support, someone can still patch your kernel. As I read the Phrack article about this technique, I shuddered again. While getting your system rootkitted is bad, SUCKIT (like LKM rootkits) might just abort your kernel if it doesn't work perfectly.

The authors of SUCKIT suggest modifying your kernel so that writes to /dev/kmem are prohibited, even to root. This will stop this rootkit, without stopping you from tuning your kernel using the /proc interface. They even suggest a one-line patch to mem.c that will do this. Some solution.

But what about stopping LKM rootkits? I mentioned earlier that there were three ways of rootkitting kernels. The third way I was alluding to works with Windows and involves installing a device driver (for information, see http://www.rootkit.com). Microsoft certainly deserves a lot of the bad marks it gets for security, but you may have noticed that Microsoft not only supports but encourages the use of signed device drivers. If a device driver has been signed, you know it has not been modified to include a rootkit and (relying on the signer of the device driver) is not a rootkit. I will confess to being less than current as a Windows sysadmin, but there was a time when someone who could administer printers could also install device drivers. And I do know that the default on XP is to make the first (and often the only) user a member of the Administrator group.

I wondered if LKM signing had been accomplished in the Linux world and found a "discussion" (a polite term for it) on an archive of the Linux-kernel mailing list. It seems that most of those involved are not interested in adding more bloat to the Linux kernel (I certainly understand that concern) by adding support for checking the signatures of LKMs before loading them. David Howell even posted patches that support checking GPG signatures of kernel modules (http://people.redhat.com/~dhowells/modsign/), but his solution appeared overwhelmed by opposition. Perhaps RedHat will decide to do this on their own for their commercial Linux version.

Proper use of LKM signing implies that any time you build kernel modules, you copy them to another system, sign them, and copy them back to the system where they will be used. As long as the signing system cannot be compromised, the signature checking mechanism will guarantee that only signed, unmodified modules get loaded into your kernel. RedHat could certainly offer signed LKMs with their distros, and those that build their own kernels could include the mechanism and the public key, in the kernels they build. Combined with disabling writing to /dev/kmem, LKM signing would appear to block an entire class of popular attacks. And it might even provide a use for the TCPA chip, in that it could hold the public key and be involved in signature checking.

I do want to add a note that FreeBSD kernels after 4.0 have the securelevel flag, which, when set to one or two, prevents kernel modules from being loaded. A positive securelevel also blocks writing to kernel memory (goodbye SUCKIT). Evil kernel modules could still be placed in a directory where they would be automatically loaded during the next reboot.

The history of computer (in)security has been one of attacks, defenses, and new attacks designed to counter those defenses. Signing LKMs could be just another failed defense. But some form of kernel defense does appear to be justified.

Anything beats corruption.

ADAM TUROFF

practical Perl

PROGRAMS TO WRITE PROGRAMS

Adam is a consultant who specializes in using Perl to manage big data. He is a long-time Perl Monger, a technical editor for *The Perl Review*, and a frequent presenter at Perl conferences.

■ *ziggy@panix.com*

Testing Web sites can be quite tedious, but it doesn't need to be. In this column I describe how I wrote a few small programs to generate hundreds or thousands of regression tests for a Web site.

One of my current projects involves reimplementing a large, dynamic Web site. Back in the go-go dot-com days, this kind of project was rather common. The preferred technique was slash-and-burn: Throw out all of the old code and implement the new Web site from scratch using whatever language, library, framework, or platform is popular this week. Along the way, the Web site would get a facelift and be upgraded to use the latest and greatest Web design techniques. The new Web site would look wonderful, garner praise, and win awards, up until the point that the cycle repeated, a few short months later.

Thankfully, the Web development scene has settled down considerably in recent years. Throwing *everything* out and reimplementing a Web site from scratch has come to be seen as not only foolish, but in many cases as not even feasible. The current Web site I am working with is the result of years of requirements gathering, design, development, testing, and debugging. It represents a slow evolution to match the application requirements with the capabilities and quirks of the Web browsers our customers use.

Throwing out years of work because some popular new system can generate Web pages "easier" or "better" is interesting but ultimately irrelevant. While the current Web site architecture may be aging and brittle, any new implementation needs to faithfully reproduce the HTML interface in use today. Customers have come to expect the site's current interface and behaviors. Gratuitous changes made for the sole benefit of the development team would negatively impact customers, and that could easily impact the bottom line. Yet some kind of change is necessary whenever the current Web site code becomes hard to maintain and difficult to extend.

Testing to the Rescue

In some respects, this problem seems like the classic "unstoppable force meets immovable object." The existing HTML must be preserved, and the easiest way to generate the existing HTML is to keep the existing Web site, however old and brittle it may be. The only way to move forward is to replace the existing Web site, but only in a manner that will faithfully reimplement the existing HTML, bug for bug. Seen this way, reimplementing this Web site is a software

project like any other, with one additional constraint, which we can check empirically as we move forward. (Fixing HTML bugs and updating HTML designs are discussions for another time and place.)

This constraint sounds cumbersome and tedious, and indeed it is. Each dynamically generated HTML page from the new site must be checked against a corresponding page from the old Web site to check that all expected content, layout, and structure is present, and *only* that material is present. Repeat this process for each of the dozens of pages to be tested. Because pages may appear differently for different users, check each page multiple times, one time for each account to be tested.

Other factors also require consideration. Like many Web sites today, this site is a front end for a very large database. The database is constantly being updated, so a Web page that passed a test yesterday or this morning might fail this afternoon because the underlying data has changed. But that kind of failure is a "false negative," since that is the expected behavior. So tests need to be updated periodically in order to ignore normal data changes, and focus on the HTML interface elements that surround that data.

As my friend brian d foy likes to point out, we're working with computers, and computers are built for doing boring, repetitive work over and over again. On this project, I need hundreds of long test scripts in order to make sure that the new Web site faithfully re-creates the output of the existing one. Rather than writing those boring, tedious test scripts by hand, I decided to write three interesting programs instead:

- find-links: Finds Web pages on the old site to examine
- build-test: Examines a single Web page on the old site and builds a test script
- MyHTMLAnalysis.pm: A module that analyzes HTML input when building and running a test script

By running two programs, I can generate a few hundred test scripts in the time it takes to get a cup of coffee. If the underlying data changes, I can just delete some tests and rebuild them while I get more coffee.

Test Setup

In order to easily compare old and new Web sites, I needed to spend a little time building an environment to support this testing activity. I started out with two identical copies of the Web site checked out from CVS, Subversion, or another source control system. The two copies of the Web site must be configured identically and run side by side on two different Web servers running on two different ports, or on two different systems. One copy will be the "baseline," running the existing code unchanged. The other copy will be the development server, where all of the changes will be made.

Having access to both versions of the code at all times is important. Whenever a test failure occurs on the development server, the baseline server should be checked with the same test script to determine whether the failure is a bug or a false negative due to normal data changes. If a test failure is in fact a false negative, the baseline can be reexamined to produce a fresh test to find real bugs in development.

With the baseline Web server in place, the first task is to find the pages to profile, the task automated by the find-links script. I could have maintained a text file of links to examine, but it was just as easy to write a program to find links for me. In the spirit of automating tedious tasks, this program emits a Makefile fragment that will build the tests.

Below is the find-links script that I used to crawl the baseline site and find all pages linked from the home page. Of course, each Web site is different, so the

rules for what to profile will likely vary from site to site. For my project, it was sufficient to look at the home page and look at any link into the site from the home page. For other sites, it might be necessary to examine a small, predetermined list of links, to perform an exhaustive traversal of every link in a site, or something in between. Note that links to other Web sites are ignored, since they are beyond the scope of what is to be tested here.

```
#!/usr/bin/perl -w
use strict;
use WWW::Mechanize;
my $usage = "Usage: $0 <baseurl> <urlpath> <testpath> [cookiejar]\n";
my $baseurl = shift(@ARGV);
my $urlpath = shift(@ARGV);
my $testpath = shift(@ARGV);
my $cookies = shift(@ARGV) || "";
die $usage unless $baseurl;
die $usage unless $urlpath;
die $usage unless $testpath;
## Specifying a cookie jar is optional
## Find URLs
my %seen;
my $number = "000";
my $mech = WWW::Mechanize->new (
            cookie_jar => {file => $cookies}
          );
my $base = "$baseurl$urlpath";
$mech->get($base);
foreach my $url ($mech->links()) {
    ## Normalize this URL:
    ## convert into an absolute URL
    ## and remove the internal anchor (if present)
    $url = $url->url_abs()->as_string();
    $url =~ s/#.*$//;
    ## Focus on links within this site, and
    ## make the URL relative to the base
    next unless $url =~ s/^$base//;
    ## Test each URL once and only once
    next if $seen{$url}++;
    ## Write out another entry in the Makefile
    my $file = "$testpath/$number.t";
    print "$file:\n\tbuild-test $baseurl $url $cookies > $file\n\n";
    $number++;
}
```

This script is invoked with four parameters: the location of the baseline server (http://localhost:8080), the URL path to the page to profile (/start), a path to deposit test scripts, and an optional file containing the cookies to use for user authentication. The location of the baseline server must be split out from the URL to process, so that build-test can create a test that assesses either the baseline or the development server.

By using this program I can create multiple test suites from the baseline Web site quickly and easily:

```
$ find-links http://localhost:8080 \
    /start ./admin admin.conf > Makefile.admin
$ find-links http://localhost:8080 \
    /start ./user user.conf > Makefile.user
$ find-links http://localhost:8080 \
    /start ./guest guest.conf > Makefile.guest
...
```

Analyzing HTML

Once find-links has run, the next step requires profiling the baseline server to build test scripts. Because these scripts will be used to check both the baseline and the development server, the location of the Web server to test should not be specified in these test scripts. I have found that the best way to specify which server to test is to place that information in environment variables, either in a Makefile or on the command line. Switching or overriding an environment variable makes it quick and easy to check the baseline server to see whether a test failure on the development server is a true bug or a false negative.

Ideally, the best way to test the output of the development server against the baseline server is to use a simple string comparison. If the output of the development server does not precisely match the expected output, the test fails. In practice, that level of rigor is simply impractical. If the output from the development server does not exactly match the output from the baseline server, it could be because one character changed or because 1000 characters changed. Also, locating where and how the two pages differ can be difficult, especially when dealing with very large HTML pages.

Furthermore, there are many textual changes that have no semantic or structural impact in HTML. The tags below are all equivalent in HTML, but fail a simple textual comparison:

```
<img src="button.gif" width="5" height="10">
<img height="10" src="button.gif" width="5">
<IMG HEIGHT='10" SRC="button.gif" WIDTH="5">
<img src='button.gif' height='10' width='5'>
<img src="button.gif"
width="5" height="10" >
```

For any meaningful comparison of baseline against development Web servers, some measure of scanning or parsing HTML output is necessary. If you are profiling an XHTML site or other XML data, you can use any of the many Perl modules for processing XML to aid your analysis. If not, then regular expressions and some of the many HTML parsing modules on CPAN can help you along.

To ease HTML profiling, it's best to put the code to analyze output from the baseline and development servers into a module used by both the build-test script and the test scripts it generates. This module contains code to do things such as find links, images, and JavaScript blocks and produce data structures that are easy to examine when building and running tests.

Purists will note that this setup adds a measure of uncertainty to the testing process. Although this is true, pre-processing HTML before testing it helpsto factor out meaningless differences and focus on the more meaningful changes between versions. Because HTML is such a troublesome format, using an analysis module provides one central place to catalog all of the differences you consider meaningless in your application.

For example, JavaScript <script> blocks *should* have a type="text/javascript" attribute. That attribute may or may not be present. The deprecated language="javascript" attribute may be present. If neither is present, browsers will assume that the content of the <script> block will be JavaScript.

Within a JavaScript block, whitespace characters are (mostly) meaningless. If two JavaScript blocks differ only in indentation, they should be considered identical. JavaScript blocks can also be wrapped with optional HTML comments. If the only difference between two such blocks is the presence/absence of HTML comments, the two blocks should be considered equivalent.

Finally, if two JavaScript blocks really do differ, it doesn't matter where they differ, just that they differ. To simplify test output, I find it useful to pre-process

JavaScript blocks and convert them into MD5 checksums. If two JavaScript blocks differ after all meaningless differences have been factored out, their checksums will differ.

Here is the function in my analysis module that cleans up JavaScript blocks for easy comparison. The analysis sounds complex, but the code is actually rather straightforward:

```
package MyHTMLAnalysis;
use MD5;
sub process_javascript {
    my $html = shift;
    ## Grab JavaScript code. Ignore attributes on the <script> tag
    my @javascript = $html =~ m{<script.*?>(.*?)</script>}sig;
    ## Normalize whitespace
    @javascript = map {s/\s+/ /; s/^\s//; s/\s$//; $_} @javascript;
    ## Remove the leading/trailing comments, if found
    @javascript = map {s{^<!—\s*(.*?)\s*//\s*—>$}{$1}s; $_} @javascript;
    ## Convert it to MD5 checksums
    @javascript = {MD5->hexhash($_)} @javascript;
    return @javascript;
}
```

Analysis functions for other portions of the HTML input are generally simple and easy to write and test on their own. HTML testing requirements generally vary from site to site, so be sure to identify what portions of the HTML input you need to analyze, and what meaningless changes you want to factor out from your tests.

Building Tests

With an HTML analysis module in place, it was time to build and run the scripts that would profile the baseline Web site and test the development Web site.

The process of building a test script was pretty simple. Each analysis function that build-test calls is mirrored with a corresponding call in the test script being generated. All of the results available to build-test are copied into the test script as test assertions using Test::More. Here is the portion of build-test that handles building JavaScript tests:

```
#!/usr/bin/perl -w
use strict;
use MyHTMLAnalysis;
use WWW::Mechanize;

my $usage = "Usage: $0 <base> <url> [cookie jar]\n";

my $base = shift(@ARGV) or die $usage;
my $url = shift(@ARGV) or die $usage;
my $cookies = shift(@ARGV) || ""; ## Cookies are optional
my $mech = WWW::Mechanize->new (
            cookie_jar => {file => $cookies}
          );
$mech->get("$base$url");
my $html = $mech->content();
print preamble($url, $cookies);
print test_javascript($html);
##...create more tests
sub preamble {
    my $url = shift;
    my $cookies = shift;
    return <<EOF;
#!/usr/bin/perl -w
use strict;
```

```
use Test::More qw(no_plan);
use MyHTMLAnalysis;
use WWW::Mechanize;
my \$mech = WWW::Mechanize->new (
          cookie_jar => {file => $cookies}
        );
\$mech->get("\$ENV{TEST_SERVER}$url");
my \$html = \$mech->content();
my \@data;
EOF
}
sub test_javascript {
    my $html = shift;
    my @data = MyHTMLAnalysis::process_javascript($html);
    my @tests;
    push (@tests, q/@data =
MyHTMLAnalysis::process_javascript($html)/);
    ## Test that all of the expected JavaScript blocks match
    foreach (@data) {
        push (@tests, qq/is(shift(\@data), q{$_});/);
    }
    ## Make sure there are no other JavaScript blocks
    pushd (@tests, <<EOT);
foreach (@data) {
    fail("Unexpected Javascript block: $_");
}
EOT
    return join("\n", @tests);
}
```

The snippet above shows how to test JavaScript blocks in a Web page. The process can easily be repeated to test more components on a Web page by adding more analysis functions to the shared analysis module, calling them in this script, and embedding the results of that analysis into the test scripts generated.

Note that this program is actually producing a Perl program (a test script), so it is important to get the quoting correct: Some variables need to be escaped because they are variables in the test script being generated. Other variables are unescaped because they are variables in build-test, where the values are being copied into the test script. The resulting program can be run using standard testing tools like Test::Harness and prove.

Finally, keep in mind that these scripts must be able to examine either the baseline or the development server. The location of the server to test is expected to be in the TEST_SERVER environment variable, and that will generally point to the development server (e.g., http://localhost:8081). When checking for changes in the database, this value would be reset to point to the baseline server (e.g., http://localhost:8080).

Conclusion

Testing Web sites is a notoriously difficult and error-prone task, but with a little advance planning and analysis, Web site testing can be a breeze. Just write a few programs to profile your Web site, and let Perl generate your test scripts for you.

JASON G. ANDRESS

IPv6: the next Internet protocol

Jason Andress works as a system administrator for Agilent Technologies. He is currently wrapping up his master's degree in CS and will soon be starting on his doctorate. He is also a recent Debian convert.

- *Jason.Andress@Agilent.com*

The follow-on to IPv4, IPv6 has not yet seen wide deployment. This article discusses the motivations for IPv6, its history, its design criteria, and some of its new features. Finally, a look at future deployment and applications is presented.

IPv6 is the network protocol follow-on to the popular IPv4,[1] the network transport layer of the TCP/IP protocol that runs the majority of the Internet. IPv6 was designed with the knowledge of all of IPv4's shortcomings and with 20 good years of experience running the Internet. IPv6 addresses the Internet's current and anticipated problems with elegant solutions.

IPv4

IPv4 was designed in 1980 to replace the already archaic NCP protocol on the ARPANET as it then existed. When first deployed, fewer than 1,000 computers were linked by IPv4. Who would have guessed that a 32-bit address space whose theoretical maximum connectivity was about two billion computers would not be enough?

Two decades after its first implementation, the explosive growth of the Internet exposed some of IPv4's limitations, the most serious of which is limited address space. The problems of expanding the address space drove the design of IPv6. IPv4 had several other problems, however:

- Its header
- Routing
- Configuration
- Security
- Quality of Service (QoS)

IPV4 ADDRESS SPACE

Two problems exist in the IPv4 address space. First, the 32-bit address does not allow sufficient address space; second, the address allocation is not granular enough. In the original allocation scheme there are five classes of addresses: A, B, C, D, and E. Of these classes, only A, B, and C are used during normal operation. These classes are broken out like so:

- Class A—125 networks, 16 million hosts per network, ~2 billion hosts total
- Class B—16,382 networks, 65,534 hosts per network, ~1.1 billion hosts total
- Class C—2 million networks, 254 hosts per network, ~508 million hosts total

NOTE

1. The effort to develop the Internet Protocol Next Generation was started in 1994 [2]. One of the fields carried forward from IPv4 was the version field. IPv4 used version number 4; another protocol, the Internet Stream Protocol, was already using version number 5. Thus, the first available version number was 6 and the name "IPv6" was born.

Note that 125 (0.006%) of the 2,016,507 networks constitute more than half of the available addresses.

One solution to IPv4's address-space problems is Classless Inter-Domain Routing (CIDR) [9]. CIDR replaces the previous A, B, and C address classes with an addressing scheme that enables the full IP address space to be partitioned much more finely. CIDR enables addresses to be assigned to networks as large as 500,000 hosts or as small as 32 hosts. The smallest block of addresses assignable under class-ful routing was 254 addresses (a class C), which was one of the contributing factors in the 3% usage rate of assigned addresses. (The other two addresses of the 256 possible are used as broadcast addresses.)

In addition to the address allocation changes brought about by CIDR, Network Address Translation (NAT) technology enables multiple systems to share a single IP address by carefully routing the combination of IP address and port number on local networks. The advantages of having a unique address for every computer on the Internet are obvious. Coupled with the proliferation of small appliances that exploit very inexpensive networking technologies, the addressing problem continues to fester.

IPV4 HEADER

The IPv4 header has two main problems that slow throughput:

- A checksum must be computed for each packet being processed.
- Each router that processes the packet must process the option field.

Unfortunately, without restructuring the header (redesigning the protocol), neither of these problems is particularly fixable.

IPV4 ROUTING

CIDR also addresses the problem with the growing size of the global routing tables. Under the previous class-ful system, the global routing tables were growing toward their maximum theoretical size of 2.1 million entries. In addition to restructuring address conventions, CIDR also implemented Hierarchical Routing Aggregation with a logically tiered structure to reduce entries in routing tables. Under this system, each router keeps only the routing information for the next routers in its own logical hierarchy. This change has reduced the number of entries in the global routing tables to approximately 35,000.

IPV4 CONFIGURATION

Under IPv4, TCP/IP-based networking requires several pieces of data to configure a network. An administrator or user must supply the IP address(es), routing gateway address, subnet mask, DNS server(s), and possibly other information. In order to simplify configuration, some networks utilize Dynamic Host Configuration Protocol servers and then enable local area network clients to request appropriate network configuration from a central server as network services are configured on that client. Although this eases configuration for the end user, it really only moves the burden to the network's administrators.

IPV4 SECURITY

The IPv4 protocol was created in an age of cooperation among research and development institutions that composed the network. The goal was to create a protocol that enabled the network to succeed; the twin notions of hostile envi-

ronments or noncooperative, even destructive, users were not strongly considered. Unfortunately, such attacks need to be taken into consideration today.

The lack of integral security in the design of IPv4 enabled the wide variety of attacks that are commonly seen today. Spoofing attacks, attacks that exploit protocol implementations to crash or disable the host or slow other connections, and a variety of others are commonplace in today's network environment.

Mechanisms to secure IPv4 do exist, but no requirements for their use are in place and no one standard exists. One of these methods, IPSec [10], sees common use in securing packet payloads. IPSec exploits cryptographic security services to provide:

- Confidentiality (messages cannot be read in transit)
- Integrity (messages cannot be altered in transit)
- Authentication (the origin of the sender is known with total confidence)

Confidentiality is provided via the use of encryption, integrity by means of a cryptographic checksum that incorporates the encryption key, and authentication by digitally signing with the encryption key.

IPV4 QOS

Quality of service (QoS) enables the priority of traffic to be adjusted to suit the type of traffic that is being handled. When IPv4 was designed, most Internet traffic was text-based. As the Internet has expanded and technology has progressed, new types of traffic such as streaming video and multiplayer gaming have created a need to prioritize traffic that is dependent on speed of delivery over traffic which does not depend on speed, such as email.

While QoS standards exist for IPv4, real-time QoS support relies both on the type of service (TOS) field and on identification of the contents of the packets. Unfortunately, the IPv4 TOS field has limited functionality and is interpreted differently by different vendors. Additionally, it is not possible to identify the contents of encrypted packets.

Design Considerations for IPv6

By 1990 it had become clear that the protocols then in use would not be able to hold up under the explosive growth of the Internet. A January 1991 meeting of the Internet Activities Board (IAB) and the Internet Engineering Steering Group (IESG) put forth five main categories as the focus for development efforts on future protocols [11]:

- Routing and addressing
- Multi-protocol architecture
- Security architecture
- Traffic control and state
- Advanced applications

Those groups completed design of the specifics of the IPv6 (then termed IPng) protocol exactly four years later [12].

IPV6 HEADER

The IPv6 header design reduces routing and processing overhead by moving nonessential and option fields to extension headers placed after the IPv6 header. The new header is only about twice as large as the IPv4 header, even with the new features and (relatively) huge 128-bit addresses. The increased header size

does not cause any appreciable delay in traffic, due to the improvements made to the header in order to ease processing.

IPv4 headers and IPv6 headers can coexist on a network, although IPv6 is not backward compatible with IPv4. A host or router must support both the IPv4 and IPv6 protocol stacks in order to process both header formats.

IPV6 ADDRESSING

IPv6 sports 128-bit addresses, in contrast to the 32-bit addresses of IPv4. This gives IPv6 an address space of 3.4×10^{38} machines, theoretically enough to assign three trillion addresses for every human on earth and 10,000 trillion other planets. However, this large space is not intended to be used in that way. Many of the address bits are used less efficiently in order to simplify addressing configuration dramatically.

Only a small percentage of IPv6 addresses are currently allocated for use by hosts, with a huge number of addresses available for future use. The address space that IPv6 provides obviates address-conservation techniques (e.g., NAT).

IPV6 ROUTING

IPv6 routing is almost identical to CIDR IPv4 routing. The IPv6 address design facilitates an efficient, hierarchical routing system that enables smaller routing tables, which, in turn, permit routing of more hosts than is possible under IPv4.

IPV6 CONFIGURATION

IPv6 supports a new stateless address configuration scheme that dramatically simplifies host configuration. In IPv6, hosts automatically configure themselves with addresses created by combining prefixes advertised by local routers with information local to the host. Even without a router, hosts on the same link can automatically configure themselves with local addresses and communicate on that local link without need for manual configuration. This new configuration system not only removes a menial task from the network administrator, but also allows renumbering of an entire network by changing local address information on the local routers [2].

IPV6 SECURITY

Compliance with IPSec [10] is mandatory in IPv6, and IPSec is actually a part of the IPv6 protocol. IPv6 provides header extensions that ease the implementation of encryption, authentication, and Virtual Private Networks (VPNs). IPSec functionality is basically identical in IPv6 and IPv4, but one benefit of IPv6 is that IPSec can be utilized along the entire route, from source to destination.

IPSec in IPv6 is implemented using two extension headers: the authentication extension header and the Encrypted Security Payload (ESP) extension header. The authentication extension header provides integrity and authentication of source, protection against replay attacks, and protection for the integrity of the header fields. The ESP extension header provides confidentiality, authentication of source, protection against replay attacks, and limited traffic flow confidentiality [14].

IPV6 QOS

The IPv6 header has new fields to define how traffic is handled and identified. By using a flow label field in the header, traffic identification enables a router to identify and potentially provide special handling for packets that belong to a flow (a series of packets between a source and destination). Because the traffic is identified in the header, support for QoS can be provided even when the contents of the packet are encrypted with IPSec.

The Urgency of IPv6 Deployment

NAT and CIDR have somewhat eased the address space issue for the current time frame; however, address space is not allocated evenly across the globe. "Some regions of the world were allocated fewer IPv4 addresses than others. The most populated part of the world, the Asia-Pacific region, was allocated the smallest amount of the remaining IP addresses: 2%, compared with 5% for the Americas and 4% for Europe. Some countries in Asia-Pacific have virtually run out of addresses already, others are close. The European Union has predicted that address space in Europe will become critical in 2005." [13] Using IPv4, China's allocation amounts to only about 22 million IP addresses. With a population of 1.3 billion people and 17 million Internet subscribers, China will shortly be entirely out of IPv4 space.

These observations, among others, are prompting many organizations to examine the transition to IPv6 in the near future. One of the largest and most visible organization with firm plans for the transition to IPv6 is the United States Department of Defense. According to a DOD memorandum on IPv6, "The DOD goal is to complete the transition to IPv6 for all inter- and intra-networking across the DOD by 2008." [6] In accordance with this memo, any network assets that are put into place as of October 1, 2003, must be both IPv6 and IPv4 capable. For those forward thinkers, there is great significance to this. If the DOD plans to be entirely on IPv6 by 2008, any company that interfaces with the networks of the DOD must be able to accommodate IPv6. This seems to be an excellent setup for a rather swift chain reaction of IPv6 conversions.

Wide IPv6 Deployment

The pain of upgrading to facilitate migration to IPv6 can be reduced by performing as much of the work as is feasible in advance. Almost all of the required changes fall into this category, making the final switchover to IPv6 an anti-climactic event. Forward-looking organizations such as the DOD are already taking these steps.

Network infrastructure changes can be phased in over time with minimum disruption by switching to routing and related equipment that supports both IPv4 and IPv6. This activity alone removes a great deal of the work in making the transition and can be accomplished fairly simply by eschewing equipment that does not support IPv6. In theory, the natural turnover of network equipment will cause IPv6 hardware compatibility to become a non-issue over time.

Much work for the transition to IPv6 involves the software and operating systems in use on clients and servers.

Almost all major operating systems have had at least some level of support for IPv6 for the last five years. Upgrading a corporate network, however, is not so simple as changing a configuration parameter and requires extensive planning in order to ensure a smooth rollout.

Migrating to IPv6 does not need to be painful, but it does need advance preparation and identification of networked applications that require major investment (vs. simple changes). Aside from making hardware and software changes, what better way to prepare for a new technology than hands-on experience?

Further Research on IPv6

Useful experience in running IPv6 networks can be gained in one of two ways: experimenting with an IPv6-based machine on the Internet, or setting up an offline test lab.

Several options exist for running a live IPv6 machine on the Internet:

- The "6bone" is an experimental Internet facility for tunneling IPv6 packets over the IPv4 Internet. See the Web site [4] to learn how to run IPv6 in general and obtain proper IPv6 addresses to use with the IPv6 Testing Address Allocation experimental protocol [7].
- Several ISPs and companies also have functioning IPv6 network connections. Connecting an IPv6 machine to the Internet poses no great difficulty given an IPv6 address, proper equipment, and a configuration [5].
- O'Reilly's *Ipv6 Essentials* [1] is a good guide to configuring various operating systems to use IPv6 and testing IPv6-oriented applications and utilities.

Appendix E in the Microsoft Press *Understanding IPv6* [3] is a guide to setting up an IPv6 test lab on Windows, including clients, routers, and a DNS server. Although intended specifically for Microsoft platforms, the main concepts translate easily to other operating systems. As specified in the book, the test lab cycles through pinging, static routing, name resolution, IPSec, and some of the IPv6 security features. Although limited, this lab setup enables experimentation without concern for security threats or other issues related to connecting to a live IPv6 network.

Practical Implications

The majority of the issues related to IPv6 fall into the categories of economics and adopter comfort level.

Economics plays a large part when looking at a migration to IPv6, not only in the sense of capital expenditure, but also in manpower, time, and other resources.

The main economic factor that needs to be considered when looking at a migration to IPv6 is the timeline for the migration. The speed of the migration can have a large effect on the cost of the project.

In the long term, migrating to IPv6 is not an expensive proposition. Over time, network infrastructure equipment will be replaced, software will be upgraded, and most of the other changes that are needed for a migration can be integrated with the normal upgrade process.

Given a very short timeline for a migration to IPv6, the cost can increase dramatically. If upgrades to network infrastructure, client software, servers, and other associated items are attempted concurrently and over a short period, not only does the cost increase, but so does the impact on users, which brings us to adopter comfort level.

The comfort level of potential users of IPv6 may not seem to be a large issue, but it definitely has the potential to be. If, during the transition to IPv6, a security issue or other problem of sufficient magnitude were publicized, the impact on large-scale migration could be significant and far-reaching. This is another case where carefully planning migrations to IPv6 can help to avert problems.

The Future

From a high-level view, the major benefits of IPv6 are its scaling and increased security. The global deployment of IPv6 will be an enabling factor in redefining the Internet as we now know it.

With IPv6, the Internet can continue its dramatic growth while embracing mobile telephones, PDAs, home appliances, automobiles, intelligent buildings, and a plethora of other devices.

Looking to the longer-term future, the ability to address and fully access any networked device has the potential to lead to new technologies and the renewal of existing ones. Consider the opportunity to take "single sign-on" a step further by addressing individuals as well as machines. An implanted chip [8] could carry an address for a particular individual and facilitate the use of cell phones, PDAs, workstations, etc. Such devices would read the user's address from the implanted chip and configure themselves accordingly. Such a technology would facilitate routing of email, retrieval of data, and other tasks that currently inconvenience users by fragmenting their data by geographical location.

Conclusion

IPv6, still in its initial stages of deployment after several years of availability, is definitely coming. It will renormalize the Internet by removing stopgap measures such as NAT, by providing a standard security mechanism for packet payloads, and by effectively removing the cap on address space. It will reduce, in the long term, the load of day-to-day administration tasks currently required just to keep networks running at a basic level, and it will relegate most network configuration to just plugging in the cable.

In the end, worry and hand-wringing over the transition to IPv6 will likely rise to the level of headline-making, but, with a little planning, the transition will be as anticlimactic as the Y2K problem.

REFERENCES

[1] Hagen, S., *IPv6 Essentials*, O'Reilly Media, Inc., 2002.

[2] Loshin, P., *IPv6 Clearly Explained*, Morgan Kaufmann Publishers, Inc., 1999.

[3] Davies, J., *Understanding IPv6*, Washington: Microsoft Press, 2003.

[4] Fink, B., "6Bone testbed for deployment of IPv6," retrieved December 2, 2004, from http://www.6bone.net/, 2004.

[5] Hinden, R., "IPng implementations," retrieved December 4, 2004, from http://playground.sun.com/pub/ipng/html/ipng-implementations.html, 2002.

[6] Stenbit, J., "Internet Protocol Version 6 (IPv6)" [electronic version], Department of Defense Memorandum, 2003.

[7] Hinden, R., Fink, R., & Postel, J., "IPv6 Testing Address Allocation," retrieved December 1, 2004, from ftp://ftp.isi.edu/in-notes/rfc2471.txt, 1998.

[8] Digital Angel Corporation, "FDA Clears Verichip for Medical Applications in the United States," retrieved December 15, 2004, from http://www.4verichip.com/nws_10132004FDA.htm, 2004.

[9] Fuller, V., Li, T., Yu, J., & Varadhan, K., "Classless Inter-domain Routing (CIDR): An Address Assignment and Aggregation Strategy," retrieved December 17, 2004, from ftp://ftp.rfc-editor.org/in-notes/rfc1519.txt, 1993.

[10] "Internet Protocol," retrieved December 16, 2004, from ftp://ftp.rfc-editor.org/in-notes/rfc2401.txt, 1998.

[11] Clark, D., Chapin, L., Cerf, V., Braden, R., & Hobby, R., "Towards the Future Internet Architecture," retrieved December 19, 2004, from http://www.rfc-editor.org/cgi-bin/rfcdoctype.pl?loc=RFC&letsgo=1287&type=ftp&file_format=txt, 1991.

[12] Bradner, S., Mankin, A., "The Recommendation for the IP Next Generation Protocol," retrieved December 19, 2004, from http://www.rfc-editor.org/cgi-bin/rfcdoctype.pl?loc=RFC&letsgo=1752&type=ftp&file_format=txt, 1995.

[13] Holder, D., "Upgrading the Net," *British Computer Bulletin*, retrieved December 19, 2004, from http://www.bcs.org/BCS/Products/Publications/JournalsAndMagazines/ComputerBulletin/OnlineArchive/may02/digitalworld.htm, 2002.

[14] Cisco, "Implementing Security for IPv6," retrieved December 19, 2004, from http://mail.cat.or.th/ipv6/sa_secv6.pdf, 2004.

ROBERT HASKINS

ISPadmin

INTERVIEW WITH VIPUL VED PRAKASH

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 Renesys Corporation, a leader in real-time Internet connectivity monitoring and reporting. He is lead author of *Slamming Spam: A Guide for System Administrators* (Addison-Wesley, 2005).

■ *rhaskins@usenix.org*

I'd like to thank Vipul Ved Prakash for taking time out of his busy schedule to answer my questions. I interviewed Vipul via email during January 2005.

RH: You are well known within the anti-spam community as the author of Vipul's Razor, as well as a founder of Cloudmark. For those readers who don't know, and to provide context to get started, I'd like you to give me some background regarding your past accomplishments. How did you get to where you are today?

VVP: During the mid-nineties, I closely followed the Cypherpunks list and was fascinated by the funkier of the cryptographic protocols that cypherpunks wrote, cited, and talked about. The notions of trust and reputation were particularly interesting to me, and I thought a lot about the mechanics of reputation networks and the problems that could be solved with them. Following a brief stint on USENET, I was suddenly inundated with spam. There wasn't much in terms of anti-spam on the Net back then, and anti-spam seemed like the perfect test application for a reputation network. That was the genesis of Vipul's Razor. At its heart, Razor is a system for assigning reputations to people who submit spam reports and, in turn, to the reports themselves.

Perl is my favorite programming language, and I've written a bunch of Perl modules that are published through CPAN. By way of merging my interests in Perl and cryptography, I've written implementations of some of the more popular cryptographic algorithms as Perl modules. I also wrote an implementation of RSA in 512 characters that was formatted to look like a dolphin. Thinkgeek carried it on a t-shirt for a while.

Another interesting project I worked on was CODD—which measures contributions to open source projects by doing source analysis to find authorship attributions. CODD is now developed at the University of Madrid, and the good folks there are doing some remarkable things with it.

RH: Can you describe how the Razor system works, and how it is similar to and different from other related anti-spam systems, namely the Distributed Checksum Clearinghouse?

VVP: Vipul's Razor is a network for sharing information about spam in propagation. The system builds a continually updating model of known spam messages, which is used by mail delivery applications to filter out subsequent deliveries of known spam.

A set of signature (fingerprinting) schemes is used to reduce spam messages to a set of signatures. Report-

ing and checking of messages are done through these one-way signatures. Reporters have an identity in the Razor system and have to authenticate themselves prior to nominating a message as spam. The back end is composed of a set of nomination servers that accept reports and a set of catalogue servers that serve the database of signatures for known spam messages.

Reports gather on the nomination side of the back end, where they are evaluated by TeS (the Truth Evaluation System), Razor's trust system. TeS examines the reports as they come in to determine the degree of agreement on whether a signature is considered "spammy" by the community. TeS also identifies the users who have reported spam in the past and whose historical decisions were mostly "unchallenged" by the community. These users accrue trust points and are eventually considered to be trusted users. It is the reports of trusted users that determine the "spaminess" of a signature, which is reflected in its confidence value. Signatures that cross the confidence threshold to become spam are replicated over to the catalogue side of the back end, from where they are "fed back" to the community.

Vipul's Razor is a collaborative classifier driven by content samples reported by its users. It predicts whether a message is spam or legit. DCC, on the other hand, determines the "bulkiness" of a particular mailing. DCC works by collating signature sightings from participating MTAs to see how many copies of a particular mailing were sent out. The task of classifying bulk into spam and desired bulk is left to out-of-bound methods in DCC.

RH: You mention trust points, scores, and confidence levels. I'd like some idea of the make-up and threshold values for some of the more common trust/scoring/confidence mechanisms in the Razor system. For example, what does a message score consist of? What score causes a message to be considered spam? What do trust and confidence scores consist of, and how are they generated?

VVP: All signatures have a confidence level associated with them; it ranges from -100 (legitimate) to 100 (spam). The confidence of a signature is a function of the number of trusted reports as well as the level of trust of the reporters who submit the reports. Razor Agents also come with a razor-revoke tool, which is used to make negative assertions—"this message is not spam." Revokes factor into the confidence as well, by pushing the confidence toward -100. TeS attempts to determine whether there is statistical consensus among trusted reporters on a particular signature. When statistical consensus is discovered, TeS selects, through a rather complex set of heuristics, a few reporters to award for participating in the reporting process. Those that disagree with the consensus are selected for penalty. An award is a positive trust point added to a reporter's trust counter, and a penalty is one or more trust points subtracted from the trust counter.

TeS is an inductive trust system. It starts out with a few trusted reporters (myself, some of the early Razor users, and folks at Cloudmark) and assigns trust to new users who tend to agree with the existing trusted users. Once the new users have accrued enough trust points to be considered trusted, they participate in selection of still newer trusted reporters. As you can imagine, over time the system becomes progressively harder to game, as the spammer has to subvert an increasing number of trusted users in order to change the disposition of their spam to legitimate. In fact, the spammer must first become trusted by agreeing with trusted reporters, i.e., by reporting messages as correctly as spam.

RH: I want to talk a little bit about the relationship between Cloudmark and Razor, but before I do that I'd like readers to have some background on Cloudmark. Can you describe how Cloudmark came to be, and what the Cloudmark products and services are?

VVP: Cloudmark was born in September 2001. I was working on the design of Razor 2, as a sudden boost in usage had put the prototypical first version at its limits. I bumped into Jordan Ritter, my co-founder at Cloudmark, on IRC. Heo was very interested in Razor and was also working on text classification algorithms for anti-spam, and he proposed that we start a company. After many whiteboard sessions to merge our visions, we founded Cloudmark with the goal of building widely deployable and highly accurate spam filters.

Today, Cloudmark provides anti-spam products to more than a million consumers and several thousand corporations. We build high-performance anti-spam engines for ISPs and large enterprises. These engines are licensed by leading mail infrastructure companies like Sendmail and Openwave. Cloudmark's SafetyBar and CEE products are Windows siblings of Razor and integrate into Outlook, Outlook Express, and Microsoft Exchange. Partner companies have integrated the SafetyBar/Razor technology in other mail products. We recently launched Cloudmark Immunity, which incorporates a proximity-based classifier for real-time online learning based on feedback from users. Immunity is designed for use in large enterprises.

RH: Can you go into some detail about exactly what anti-spam checks are in the various Cloudmark products? Does it use Razor data or a separate data set? Also, please talk more about Cloudmark Immunity and the "proximity-based classifier," as I am not sure what that is.

VVP: Cloudmark SafetyBar and Cloudmark Exchange Edition products plug into the same network as Razor and use the same data. The difference is that they support more signature schemes and perform with better accuracy and precision. Razor Agents provide accuracy to the order of 90%, whereas SafetyBar and CEE record 98% or higher with very few false positives. In fact, in the last four tests conducted by *PC Magazine*, SafetyBar was the only product to record zero false positives. Cloudmark's enterprise and ISP products are based on homegrown classifier technologies, but are trained from the data set created by Razor. Over a million people report spam to Razor, and the reports are verified by TeS, which results in a high-quality, comprehensive database with which to train our classifiers.

Immunity was designed to solve some of the problems inherent in statistical classification systems such as naive Bayesian, which, while providing compact hypotheses, were not designed to work in adversarial and rapidly evolving environments like anti-spam. Immunity maps its training set in a hyper-dimensional space (such that a spam is a point in this space) and classifies incoming documents based on the disposition of points that fall in its neighborhood. Unlike Bayesian classifiers, Immunity's classifier can be trained on one-off samples, can modify its model based on individual pieces of feedback (since accepting feedback is a simple matter of making an entry into the hyperspace), and can provide filtration that is specific to individual users. It's also more robust in that it is not vulnerable to common poisoning attacks.

RH: Some people have criticized you/Cloudmark for using the Razor spam signature data for commercial purposes (i.e., in Cloudmark's products). Is this a valid complaint? How do you respond to this criticism? What have you done (if anything) to help mitigate the issue?

VVP: We pour the data submitted by Razor users and by SafetyBar users into the same funnel and it all goes through the same trust system. Both communities benefit from the collective reports, so they actually do better than they would have without each other.

When we founded Cloudmark, there was concern about Razor Agents disappearing or moving entirely into the commercial realm. Such concerns have now

been alleviated, as we have done a major release of Razor Agents almost every quarter since and continue to add hundreds of new Razor users every day.

Merging open source and commercial software worlds is hard, especially in the face of extreme ideological commitments that people make about developing software. It has been an interesting experience for me in that regard, and I think we've done a decent job of it.

RH: I'd like to get your ideas specifically about the area of reputations in combating spam. [For background on anti-spam reputations, please see my article in the February 2005 issue of *LinuxWorld* titled "The Rise of Reputations in the Fight Against Spam."] Can you give readers some idea of how reputations have changed the fight against spam? How have spammers changed their tactics in response to reputation-based systems? I'd like to hear your views not only as they apply to Razor/Cloudmark, but the other reputation services as well—Verisign, Ironport, Kelkea, etc.

VVP: I believe reputation systems are central to the fight against spam; I like to think of a reputation system as a predictive model that uses a combination of historical performance and opinions of trusted sources to make a good/bad prediction about an unknown object. The objects evaluated in the context of email are usually IP addresses, sender domains, individual or institutional senders, email content, and newsletters/mailings.

Reputations require identities. Since email, as a protocol, is mostly identity-free, identification mechanisms have to be overlaid. SPF, Sender-ID, and DomainKeys, three schemes that are garnering a lot of support, attach identity to sender hosts and domains via reverse DNS lookups. A few reputation systems are cropping up to assign reputations to domains and hosts as identified by these schemes. Cloudmark Rating is one of them. As I mentioned earlier, we also use reputations in the context of Razor. From that perspective, Razor's signature schemes can be thought of as a way to assign identity to spam content.

In general, reputation systems are good news for good guys. Once you can identify yourself, recipients can ensure delivery. Legitimate communications can pass through anti-spam mechanisms without evaluation if identity and reputation can be established.

Reputations are changing the playing field, i.e., the anti-spam problem is moving to a slightly different place. There are two kinds of attacks spammers can mount against reputation-based systems. The first is to avoid identification. As long as identification systems are not pervasive, spammers will try to blend in with "unknown" mail. To fight SPF/DomainKeys, spammers will register a lot of domains and cycle through them as soon as a domain is considered spammy. In response, reputation systems need to be quick, allowing only a little spam to get through per domain. If the system is fast enough to make the cost of domain registration prohibitive, then we have a good defense against this attack.

The second, more ominous, attack is to hijack the mail infrastructure of senders with good reputations. Spammers are doing this today through zombie networks. The current zombies don't exploit SPF/DomainKeys, but it is the obvious next step to infect machines inside a good network, look up advertised MX servers, and inject spam through them. This is a hard attack to defend against, especially with identity-based methods, which assume perfect internal security. To provide meaningful filtration, reputation systems would have to learn the patterns of abuse associated with hijacking attacks. Personally, I believe the solution to this attack would come from content-based methods, something along the lines of raising a "content-exception" so that reputation systems don't persecute abused domains, while still disabling the propagation of spam.

I am not very familiar with the internals of other reputation systems. I believe Verisign and Habeas are providing accreditation (as opposed to reputation) services, where they do out-of-band research and accreditation of good senders.

RH: Each anti-spam vendor seems to have its own idea of what a reputation is. Obviously, this poses a major stumbling block for both anti-spam vendors and spam-fighting system administrators. Anti-spam reputations are often compared to credit bureaus, except that the risk is the "spamminess" of the sender rather than someone not paying their bill. Are we going to have a common definition of a reputation the way credit bureaus have? Failing that, will there be a standard API so that there can be one programming interface to multiple "reputation bureaus"?

VVP: There is considerable value in a standardized reputation API, but we need to be careful about representing the semantics intended by individual reputation systems. Reputations are measurements of behavior against a defined policy. In a space where different vendors are measuring very different aspects of email, it becomes necessary to understand exactly what they are measuring and how these measurements map to a filtration policy.

Many ISPs have their own reputation databases based purely on the quantity of email received from senders,\; others measure conformance to their AUP; yet others assign reputation by co-relating senders with the nature of their emails' content. Cloudmark's reputation system is based on reports from the trusted members of its community of users, whereas Habeas is based on real-world identity and mail-stream permission levels. Almost every reputation system out there is different.

One option for standardization is to allow the user to ask contextual questions of a reputation service so it can offer a binary decision (accept/drop) to the user. The other is to find a way to encapsulate the diversity of semantics so that the burden of finding the appropriate policy is shifted to the user of the reputation system.

RH: You mentioned SPF and DomainKeys; recently in the open source arena we have seen lots of attention on the area of reputations as well as domain authentication (GOSSiP and Sender Policy Framework/Aspen, among others). What impact will open source anti-spam reputation applications such as GOSSiP and SPF have on fighting spam?

VVP: GOSSiP combined with SPF and DomainKeys has the potential to enable private, quasi-disjoint, peer-reviewed email networks. It will be very exciting, especially when GOSSiP or a similar distributed reputation scheme achieves critical mass. The cost of joining the email network would increase selectively for spammers, but good guys would be able to get "introductions" to the network. I also think augmenting protocols is best done as open-source projects. Since adoption is the biggest hurdle these schemes face, publication under liberal licenses helps immensely.

HAL MILLER

customer requirements specifications for system administrators

Hal Miller, having returned from three years as a recalled reserve officer (Lt. Col., U.S.A.F.), is in the UNIX Infrastructure group at BankOne in Ohio. In a previous life, he was president of SAGE, and before that, president of SAGE-AU, as well as a "working sysadmin."

■ *halm@sage.org*

Whether you are trying to find your way around a "new" site, reviewing for a potential rework of your present organization, or planning for a new project, sometime during your career you will probably be faced with having to pin down a set of formal customer requirements. Informally, you face it every day, but most of us deal with that "by the seat of the pants." Formal specification is much more involved and takes some planning in itself.

This article discusses the issues involved, purposes of the specification, goals, and roles. It is intended to be a guide to those sysadmins who suddenly need to fit this process into their already busy schedule.

We like to think that we are continually observing, understanding, and meeting customer requirements as they occur or effectively anticipating and preparing to meet those requirements before they arrive. As a rule, that is just wishful thinking.

In the world of "firefighting" that describes a typical sysadmin's workday, such planning is pretty thin, and the time needed to plan around customer needs is just too costly to spend regularly. Thus, periodically we should consider tackling it as a priority project. That periodicity will depend on many things, such as how often we meet with customers already (in an appropriate forum, as opposed to individually at their desk), the nature of the business, and the size or complexity of the organization.

The Customer Requirements Specification (CRS) is a document normally used for computer systems support planning purposes. It identifies in some depth issues the customers have now and foresee for the next period, issues the computing support group foresees, and any potential business climate changes in the works.

The CRS creation process involves four classes of people: "customers," those who use the computing environment to perform their own daily tasks; the "support team," responsible for the design, implementation, and management of the computing environment; "management," responsible both for approving budgets and for validating the requirements given by "customers"; and "vendors," who can supply information on available technology and tools that might be applied once the CRS is completed.

A CRS should point out to the computing support team any gaps in current coverage, any "sore points" as seen from the customer's vantage point, and holes

in current planning. It should reassure the customer about the support team's level of interest and ability to meet perceived needs, and the support team about their ability to prevent contention with customers and last-minute emergencies. It should also become the primary input in short- and long-term planning, both for budgeting (including personnel and organizational issues) and for infrastructure design work. The CRS is not an audit—that's a different process with a different set of goals. The CRS is a snapshot of thought, not implementation. It is important to avoid building the environment first, then trying to justify it by weighting the CRS to fit the design!

Lay of the Land

You will need an organizational chart that maps not only levels of official management, but all those who have an impact on budgeting and staffing plans. Begin by asking for the official organizational chart, then modify it to suit your goals, asking employees at all levels for their input. This process may have value beyond your immediate needs: the completed chart may highlight problems in the current organizational structure and thus perform an invaluable service for management as well.

Once you map the organization, assemble an accurate inventory and description of "what is." List physical facilities along with the business purpose of each; current computing facilities, including the space, power, and bandwidth available; and numbers of personnel supported at each facility. If users at a given location fall into multiple categories (e.g., heavy programming, administrative staff, and CAD), chart those uses and their impact. Determine the extent of inter-site computing and managerial relationships, storage currently in use, types of operating systems, software packages, and numbers of users of shared areas at each site.

Take a look at the management structure. Who actually makes decisions, considering both those in authority and those supplying options and recommendations? What process is used to determine what should go into the budget? What parts of the whole cost of doing business go into each manager's piece of the budget? Are personnel costs included in the sysadmin portion? Does that include training and conference attendance? Medical benefits? Team t-shirts (don't underestimate the value of team-building toys)? What is the information flow during the budgeting cycle? Who assigns tasks and responsibilities? Is there a reasonable delegation process, or is authority closely held (read: micromanagement)?

Look at the IT Team. It should have some identifiable structure, even if "flat," "chaotic," or "harried" seems to apply. How does information flow to, through, and from the computing support folks? What is the current backlog of "repair" type tasks? Are larger-scale projects getting done? Are they even on the drawing board, or have people given up? What are the current problems—perceived by IT folks, management, and customers—with the IT Team? Are any changes anticipated? Are any agreed upon as being required even if not scheduled?

When you have a handle on the current organizational structure and situation, turn to the business itself. What is the true purpose or type of business? Is the organization there to conduct retail sales? Manufacturing? Supplying services? If a clear answer to this isn't in everyone's mind, something needs fixing right there. Don't be afraid to ask the marketing folks, whose job it is to deduce these items with pinpoint precision.

What sort of business model is followed? How does the organization run—what are the processes and practices? Try to trace on paper, step by step, the normal flow of information throughout the organization, bearing in mind that (1) it probably won't match the official line and (2) there may be many different flows for different purposes.

You might find some interesting "We've always done it that way" and "I just ask Joe" situations. What use is currently made of the computing power in place? Fancy typewriters or actual compute engines? Are they being underutilized, or overburdened for the tasks currently assigned? Are changes in business process being anticipated? Most important, how willing is the organization to review those processes?

When an inspection, review, or consultation is announced, people have reactions and expectations. Do customers feel threatened by the CRS? Do they have axes to grind and see this as a way to get their views heard, perhaps anonymously? What are customer expectations of the computing environment, and are those expectations being met? Are they knowledgeable? Are they even reasonable? Is there some change underway that may alter those expectations? Is there fear that the CRS might get someone fired or disciplined? Is there push-back, and, if so, from whom?

What other gains might there be from the CRS? What are the IT team's expectations of it? Are there affiliated organizations that may need to be interviewed? Suppliers, dependent groups, potential competitors, etc., might have something of value to add.

When you have a good picture of the expectations and a good idea regarding any process reengineering you may need to tackle, goal-setting becomes important. The target becomes a refined set of goals and agreed-upon deliverables.

Two sets of goals will dominate: those of the CRS performer (you) and those of your customer organization, including all of its various parts. The CRS performer should work toward:

- obtaining the information required to recommend computing environment design;
- obtaining information required to size equipment;
- making projections and establishing what the growth patterns are;
- determining support requirements;
- pinning down costs, both up front and recurring; and
- performing process engineering for business and information flow.

The customer will supply a set of goals. Get them in writing; in fact, help them to draft the list. Make each goal clear, achievable, measurable, and focused (each with a single purpose).

Your deliverables include:

- A survey questionnaire
- All the raw data from the survey
- A summary of the reduced/refined data
- A final report

The report should cover projected growth, proposed process changes, proposed computing environment design criteria, a draft overall computing environment design, and a ball-park estimate of up-front, recurring, and personnel costs for the new computing environment as proposed.

Your Knowledge Target

Creating the set of deliverables requires a lot of learning and processing of organizational information. This section discusses the knowledge that will be chased down and how to target it.

A typical analysis of computing needs begins by examining the current systems. How many files are stored and in what sizes, types, and hierarchical layout? To whom do they belong—are they grouped somehow? What sort of file service is in place? Does it cover all machines, or just certain subsets? If the latter, is it by

design, by default (grew that way, or nobody bothered figuring out how to implement something more), or by technical limitation? What types of servers are in place? What types of client machines, operating systems, and applications? Where are they run—are they grouped, and if so, why? Look constantly for things that grew without design or things that should have been cleaned up and just never were.

What numbers, types (including experience levels), groupings, strengths, and weaknesses do you find in the user community? Do some tend to drive requirements more than others? Why? Do some tend to drive priorities more than others? If so, is it by authority or just by being the squeaky wheel?

Analyze the existing IT team. How is it organized and what are its strengths and weaknesses? Is training required? Training will need to be addressed later with regard to the new environment, but holes may already exist that need to be filled regardless of what new computing technology is applied. Include a full-scale, critical review of the non–IT team people in the organization who have root or administrator privileges. Emphasize that these people have the same responsibilities as the regular support staff.

Document the existing computing environment. Include a detailed inventory that covers the networking infrastructure and security implementation as well as servers and desktops, remote capability (location-independent computing, dial-in, etc.), and written policy and procedure. Analyze the help-desk function.

Determine and document your opinion on the adequacy of the environment (and its parts) for the current requirements and processes, including the perceived requirements of various groups in the organization. Carefully review the security status and requirements of these groups. This will be critical later when you try to justify the security implementation you will recommend.

In fact, drive hard on the security issues, noting the number of ways, posted on the Internet, to break into the organization's machines/devices/OSes/whatever. Carefully document the access requirements of those outside the organization; too often people "out there" collaborating with your organization on something demand completely open access, get it, and are not protected. Find all such instances, back doors, etc. Document existing location-independent operations and measures of actual usage.

The next step is to figure out where your information is hiding. Who has it, and is there hard-copy documentation? Electronic documentation is handy for general reading, but if it's on disaster recovery, it's hard to read it when it's most needed. Who has the corporate history for the computing environment, and what chain of control has been used to pass things along? What things are left as is because "things might break" if they're changed?

For all the items above, determine how best to obtain the information: through direct questioning, by probing servers, or by examining fielded configurations.

Writing the Survey

In those areas where people are the best source of information, compile a set of survey questions. Don't be afraid to use a template or gather examples from associates who have done this before or from sites surveyed in the past. There is nothing wrong with plagiarizing questions.

Begin by breaking down the knowledge target issues (see above) for which information is not available. List specific information that needs to be gleaned. Make the list very comprehensive—don't worry about the number of questions or the number of people to be surveyed.

Craft the questions carefully to get directly to the required piece of data. Try hard not to display biases here; avoid telegraphing the preferred answer in the question.

Target the question sets to specific audiences—there's no need to give exactly the same survey to everyone; fairness is not an issue. Bound the questions: instead of "Do you purchase software regularly?" ask "In the past six months, how many software packages have you purchased?" Be exact and avoid ambiguity. Instead of "Are you experienced in installing hardware?" ask "How many internal disk drives have you installed in PCs this past year?"

If you cannot extrapolate accurately from those answers, ask secondary questions, such as adding, "How many PCI video cards?" to the previous example. If you give multiple-choice questions, make the alternatives clear and show no leanings or biases. People tend to try to give the answer they think you're looking for, e.g., if there is an "all of the above" and each option is reasonable, people may tend to use it, meaning data may either be skewed or too weak to mean much.

Ask different questions aimed at getting the same data: "How much time do you spend per week doing X?" and "Order the following based on the amount of time you spend on each," then cross-check them to validate as well as to build on your knowledge.

Ask some similar questions (perhaps in different forms) of members of different groups, looking for perceptions: "Do you answer all highest-priority trouble tickets within established time guidelines?" and "Did the IT team meet your highestpriority trouble tickets within within established time guidelines?"

Ask both the customers and the IT team members to order some sample tasks based on priority, to find out whether there is agreement and comfort with the prioritization process. Ask at least one text question that gives people the chance to express their opinion on overall issues. Obviously, you should check spelling, ensure sentences are complete (or use the "complete this sentence" approach where appropriate), be grammatical, be certain of your use of vocabulary, and make sure to use words in their most commonly understood meanings. Remember, simple words are fine when properly used.

H, have someone proofread your survey, both for language use (including typographical errors) and for ease of understanding. Discuss with your reader what they got out of each question, to make sure the respondents will be likely to provide the desired data.

Administering the Survey

As noted above, it is easy to bias a survey. The most common ways to invalidate results are to:

- Incorrectly select the participants.
- Be less than fully objective in your language.
- Structure questions in an unclear or ambiguous way.
- Mistime the administration of the survey.

The middle two were covered above; the first and last are addressed here.

Many factors influence how people respond to questions. One significant factor is the participant pool's other members. People tend to try to maintain a consistent reputation for themselves within a group. If the group they are in seems homogeneous based on some given factor, the answers will be homogeneous if questions revolve around that factor. If the group is very diverse, people may feel more free to give independent opinions.

Some factors to consider in selecting the participating audience include the size of the group, social factors, time of day, and the physical setting. Will the survey be administered to a large room full of people, small groups together, or individuals? Will each hear or be heard by others? Will responses be treated as anonymous or confidential? If grouped, will the groups be homogeneous or heterogeneous? Are managers and subordinates, genders, races, etc., combined or segregated? Are only members of certain groups selected, e.g., upper-level but not lower-level staff? Will the survey only cover day shift, excluding night folks? Will people take the survey home or otherwise work on it off-site, or will it be completed during work hours? One session, or multiple sessions? All of these affect the results.

The number of questions on a survey has some bearing on the accuracy of the results. Too few questions may yield too little information, but too many may yield contradictory, or, if the participant tires of responding, poor information. Decide exactly what information is desired and construct your questions to get that information. Be prepared to give out different sets of questions, in multiple combinations, to employees in different situations.

Data Reduction, Refinement, and Analysis

Once the information has been gathered, how is it processed? What does it mean? Reducing data from raw form to something structured, refining it to address the target knowledge list, and then analyzing the results is the whole reason for the survey process.

Begin by setting the knowledge target list on the desk in plain sight. Continually refer to it to keep on track. As the saying goes, when you're up to your knees in alligators, it's hard to remember that you're there to drain the swamp.

As much as it may be of interest to go down any side tracks that appear, just make a note to get back to them another time and press on. Otherwise the process will take forever. The exception, of course, is if you run into something so significant and so unexpected that it completely changes the purpose of your project, such as an organizational decision to change from manufacturing automobiles to selling hot dogs.

Good questions will generally yield some reasonably quantifiable responses, at least in most cases (some things are better asked in non-quantifiable form). Tallying responses is a fairly easy form of data reduction, and percentages can quickly be calculated. Of course, the wording of a question may turn out to be less clear than had been hoped. Responses may then fall into two or more groups: those who thought the question meant one thing, and those who thought differently. In other words, you may actually be looking at answers to multiple questions.

Solutions to this problem could involve resurveying, throwing out questions, or guessing. This is a good time to recall that surveys are not an exact science, but a picture of people's opinions based on their understanding of what is being asked. Don't rely on the results as "fact," but merely as indicators for planning purposes.

The hard part of data reduction is keeping nonquantifiable responses in context as part of the whole viewpoint of the participant. Many textual answers tend to get off the point of the question, and may be misleading if not understood within the framework of the rest of that individual's answers. Don't assume it's possible to cut and paste all the answers to question 37 into one large file and make complete sense of it—critical issues could be obscured by doing so.

Take the time, when reading a text answer, to glance through the quantifiable answers to relevant questions on the same survey response, to ensure that you

understand the participant's full intent. Yes, this is time-consuming, but it is important, especially for responses that cannot be tallied easily.

Once the data has been reduced, it's time to refine it. Raw data is numbers and text strings. Reduced data is effectively a summary of those numbers and text strings. Refined data is information. Analysis, which comes shortly, is the interpretation of that information.

Refining quantifiable data might include selecting a value-added method of presentation. To some, that's a chart or graph; to others, a list of percentages. Determine the preferences of the audience for the final report and try to accommodate their wishes.

For some of the process, you yourself are the target audience, so you get some input too! Refining this kind of data again includes throwing out questions, rephrasing, and asking againas necessary. Take whatever measures are required to give the results meaning. Regard this step as a reality check.

Refining unquantifiable data is a more difficult task but has the same goal. Presentation is more difficult, especially to managers who tend to base decisions on trends, budgets, or percentages. Some information doesn't fit neatly into numerical form, however. Bear in mind the most important points to be conveyed to the target audience, and try to see the presentation through their eyes—might they misunderstand any of these points? jump to a conclusion without seeing what is really there? This step doesn't deal with the results of the survey, just with ensuring that whatever the results were will be conveyed.

Once the summarization, presentation, and reality checks are done, the analysis phase commences. If the knowledge target list was clear and valid, the questions simple and to the point, and the data collection methods trustworthy, the refined results should neatly fill in the blanks on the knowledge target list.

If they don't, it is your job to decide whether you should interpolate, guess, or try again. The actual analysis varies dramatically depending on what the scenario provides and cannot be covered here in any depth. Don't get bogged down in numbers or semantics—the information is based on opinion and survey and is not "proven fact." Use it to gain an understanding of what the real needs are, with the purpose of determining how best to plan to meet them.

A good CSR has plenty of built-in flexibility, since organizational needs change constantly, so aim for being reasonable and appropriate, as opposed to being right in any absolute sense.

Final Report and Follow-on Proposals

Once the data analysis is complete, it's time to assemble a report of the customer's requirements, along with proposals for meeting them. *This is the set of deliverables.*

During the process of preparing this report, keep the sheet describing the agreed-upon deliverables in front of you. Sometimes it is a good idea to structure those deliverables as a set of questions, at least for the purpose of writing the report—this gives a clear structure for responding to the customer and serves to remind you of which issues have not yet been addressed thoroughly.

The final report should briefly restate the goals and deliverables and then quickly review the process followed to achieve those goals and deliverables. The final report should be a reasonably short paper (effectively an executive summary) that summarizes your findings and recommendations in high-level, rather generic terms. The reader should be referred to appendices for both raw and refined data, with detailed findings and recommendations in additional appendices.

Appendices should include the following:

- Survey questions
- Raw survey data
- Refined survey data, in various presentation formats
- Findings and expert opinions
- Various options, including pros and cons, costs, and proposed timelines
- Recommendations

The findings are really the meat of the report. This is the assessment of the actual customer requirements, enumerated by category—CPU server power, storage space, reliability issues, network bandwidth, etc.). Explain what is needed, with occasional references as to why (but not in depth). Be prepared to defend those findings orally, and possibly later in writing, to those who will read your report.

Remember, you are the expert here, and the customer has asked for your opinion: render it. Two rules apply: professionalism, and the KISS Principle. Gear the CRS to your target audience, but even if they're computer-savvy, it's preferable to focus on the big picture rather than getting bogged down in detail.

If you are supposed to do additional consulting, provide proposals at this point that address your recommendations. For instance, you might be suggesting that you build out the new computing environment you're recommending, or that further investigation/research be undertaken for business process consulting. Proposal documents should be separate from the final report—you want your report accepted by the customer as a completed consulting task, regardless of whether or not they accept your proposals.

Periodic Reviews

One of your recommendations is likely to be a periodic review or performance of this sort of CRS. Needs change, organizations change, people change, technology advances, and (over time) the purposes for which a computing environment exists may have changed sufficiently to make the existing one obsolete or irrelevant. In any case, the organization will benefit from regular review to ensure that the original requirements specification was valid—remember, we're working with a survey of opinions here; relying on people's thoughts of the day is an inexact science.

Write your CRS, including goals, explanations (e.g., why you grouped the participants as you did), and recommendations, with a review in mind. Assume that someone else will conduct the next one. Even if it's you, it'll likely have been long enough ago that you will have forgotten your reasoning for whatever isn't written out in detail.

Appendix: Examples

KNOWLEDGE TARGET LIST

This is a sample list of information to be considered during the drafting of recommendations. Anything not known is a candidate for survey questions. A box (■) indicates "critical"; a dash (–) indicates a normal level of importance.

- ■ Purpose of the business/organization
- – Purpose of the computing environment
- – Expectations of the computing environment

- Current information flow in the organization (not specifically computing-related)
- Projected information flow after a new computing environment is implemented
- Disaster recovery and business resumption models and plans
- Plans for hardware and software migration, life-cycle replacement, legacy issues
- Perceived problems to be resolved by a new design
  - Current network layout
  - Current computing policies
  - Current security implementation and policies
  - Who drives actual priorities for support work
  - The organization of the existing support group
  - Physical plant layout, number of floors and buildings, remote sites related
  - Number, size, and interrelationship of sysadmin groups involved; their depth and abilities
  - Number of current servers, OSes, manufacturers, models, hardware configurations (cards and card layout, space, cabling, OS versions, usage), life expectancy
  - Number of clients, purposes, integration requirements (AppleTalk, NFS, Samba, printing), file-sharing requirements
  - Applications in use, versions, predicted changes in patterns
  - Quantity of storage applied, models, brands, amount in use, usage patterns (e.g., periodic, long-term, number of readers/writers, aging process), data types, rates of change of files, databases, raw vs. cooked file systems, filesystem specifics (sizes, chunks), RAID levels, snapshot usage, mirrors, types of service (NFS, CIFS, etc.)
  - Number of users (on-site and off-site), external connectivity issues, collaborators or sharing issues, location-independence requirements, impact on security
  - High availability requirements, uptime
  - Software installed, license issues, version issues, restrictions
  - Computer room construction, availability of resources (power, space, air conditioning, cabling, security)
  - Number of servers accessing the same data
  - Business hours, maintenance windows
  - Interoperability issues between groups, sites, machines/OSes
  - Tools in use to manage site, monitoring, logging
  - Centralized vs. decentralized services: printing, email, servers, management reasoning
- Applications to be used in the new environment
- Growth expectations, storage, compute power, users, reasoning
  - Off-site connectivity requirements upcoming, collaborations, “shared” areas outside the new environment
  - Backup plans and intentions, off-site and on-site mirrors, reasoning
- High availability and uptime requirements predicted, reasoning
  - Hardware and software vendor biases, reasoning

- Possible changes to the business model, business purposes, collaborations
- Software version requirements anticipated, potential conflicts, support issues
- Support group plans
- Other vendors involved in design phase, implementation phase, internal people involved
- Corporate stability, financial status, funding for this project, budget model timeline
- Plans for training support group and users
- Support conflict managements, SLAs
- Special requirements, e.g., 10-minute snapshots, root/administrator access for users, high performance CPU crunching

SAMPLE SURVEY

INTRODUCTION

It is always a good idea periodically to review the computing environment to ensure that it continues to meet the needs of the user community. When resources are stretched and need to be expanded, or when significant change to the user community is in process, this review becomes even more important.

The organization is out of IP address space and is short of disk space. The backup process leaves significant gaps in coverage. Funding changes are on the horizon. The current computing environment grew in a piecemeal fashion over a period of years, from a time when the needs of the organization were different. Although this is a common situation, this environment needs review and possible reorganization for effectiveness and maintainability. This survey is designed to lead to a Customer Requirements Specifications document which will provide the basis of a plan for computing environment changes.

The CRS document will help define the services to be provided. Your timely participation in this survey is absolutely crucial to its success and will lead to an improved computing environment. Please feel free to ask questions or to make comments in the margin. Answer as many of the questions as you can. Thank you for your cooperation.

PEOPLE

P1. What is the name of your group or project?

P2. Who are the people in your group?

P3. What are their usernames?

P4. What computing equipment preferences does each have?

P5. What actual computing equipment needs does each have?

P6. How are these people divided into subgroups?

P7. What permissions does each person need with regard to files outside of their home directory?

P8. What permissions do the other members of each group need with regard to files in home directories?

P9. Who else, outside of the group, needs permissions in your project and/or home areas?

P10. What computer-use training does each person need? Who provides this?

P11. How many new positions are scheduled to be added to your group? When?

EQUIPMENT

E1. What computing equipment belongs to your group?

E2. What computing equipment do the members of your group use?

E3. What is the age and end-of-life projection for each item?

E4. Is each suited to its current tasks?

E5. Is each suited to upcoming requirements?

E6. Will any be transferred to another group or organization?

E7. What are your "uptime" requirements? Why?

E8. Are there periods where you are able to accept "scheduled outages"?

E9. How long would you be able to withstand a "catastrophic outage"?

E10. How much data do you have online now?

E11. What change do you anticipate to that level of data online? What rate?

E12. What performance changes do you need? Why?

GRANTS OR OTHER FUNDING

G1. What grants are current in your group?

G2. When will each run out, in its current iteration?

G3. What is the likelihood of each being extended? For how long? When?

G4. What equipment does each grant own? Are there other co-owners as well? Co-users who are not part of the grant?

G5. What ties does each grant make upon equipment, in the sense of funding?

G6. Will any grants be transferred to another group?

G7. What additional grants are being considered? When?

G8. What additional non-grant projects are being considered? When?

SECURITY

S1. What collaborative ties do you have to organizations outside of your group, but within the overall organization?

S2. What collaborative ties do you have to external organizations?

S3. What collaborative ties do you anticipate being formed? When?

S4. How many of your group use location-independent computing? (This includes dial-in or remote login from other dial-in services or from other sites.)

S5. What services do you and your group use remotely?

S6. What use do you make of superuser/administrator access on your machines now? Why? When?

S7. Are there parts of your data that require extra protection?

SOFTWARE

O1. What software packages does your group use? Please give the version number of each.

O2. Where do you get those packages?

O3. What other software packages will you need? When?

O4. What operating systems does your group use now? Why?

O5. What changes do you anticipate in your operating system needs? Why?

MISCELLANEOUS

M1. What current computing needs do you have that are not being satisfied?

M2. Do you have special needs for additional services?

M3. What current computing problems do you face?

M4. What other computing requirements do you foresee?

M5. What responsibilities do you see as belonging to the support team?

M6. How do conflicts in requirements or priorities get resolved?

NICK STOUGHTON

update on standards

Nick is the USENIX Standards Liaison and represents 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*

At the end of last year, the USENIX Board of Directors asked me to prepare a report for *;login:* on the activities in the world of formal standards in 2004.

And a very busy year it's been.

For two years or so, since the Austin Group revised the POSIX standard completely, the majority of activities have been centered on maintenance. Maintenance is a good thing; the fact that since 2001 we have addressed hundreds of problem reports against the 3700+ pages of POSIX, as well as publishing two Technical Corrigenda, shows that the standard is being used. People are reading it carefully and asking hard questions of the form "Did you really mean to say this?" Sometimes the answer is "yes," sometimes it's "no," but every question is carefully reviewed , and if necessary new text is written to clarify the meaning.

Having the standard freely available online (see http://www.unix.org/single_unix_specification/) helps enormously in spreading the use of POSIX: no system implementer or application developer can excuse nonconformance with "I couldn't afford the standard." Additionally, the relative ease of licensing the text means that open source implementations have started to adopt the actual standard text for their man pages; both FreeBSD and the Linux Man Page project have licensed it for this purpose. Increasingly, open source solutions set POSIX conformance as a deliberate objective.

POSIX is about to embark on its next major revision. This year will see the development of material suitable for inclusion in POSIX as Open Group specifications, IEEE specifications, or ISO Technical Reports. Then, during 2006, this new material will be worked into the core document and published, probably in 2007. Work here includes fixing some of the more controversial items from the maintenance process, adding new interfaces widely used in GNU utilities but missing from POSIX, and handling convergence with other ISO standards (see the LSB discussion below).

The Linux Standard Base

POSIX is all well and good for people working in software at the source level and looking for portability. However, there is still a substantial market for closed source applications. Although Linux has undergone a meteoric rise in popularity over recent years, the lack of proprietary software is still seen as a stumbling block for enterprises that equate "proprietary" with

"commercial." The Free Standards Group (FSG) has been developing a very different sort of standard to help address this problem: the Linux Standard Base (LSB). The LSB is an Application Binary Interface (ABI) standard, whereas POSIX is an Application Programming Interface (API) standard.

The LSB ABI allows an application developer to build an application in such a way that the binary will run in the same way on any conforming implementation. You, as an application developer, can feel sure that your conforming application will run on any conforming distribution. The ABI deals with issues such as versions of libraries, the value of certain macros, and so on. Much of the detail of what a particular interface does is left to some underlying specification such as POSIX.

Although the LSB is heavily based on POSIX, with most of its interfaces (excluding C++) coming from that source, there are a few notable differences between some implementations of these interfaces. In particular, POSIX may specify one behavior, but glibc or some GNU utility has implemented a subtly different behavior. POSIX is in a position to dictate how a conforming implementation should behave. The LSB, on the other hand, is an ABI standard, describing how something has been implemented rather than how something should be implemented.

A good example is the open system call. POSIX states clearly that open "shall fail if . . . [ENXIO:] The named file is a character special or block special file, and the device associated with this special file does not exist." However, the Linux kernel returns ENODEV in this case, rather than ENXIO. If you are writing an application that cares about the value of errno (i.e., tries to perform some sort of error recovery for certain values of errno), then you'll need to test for both values for this case.

One of the goals of the next POSIX revision is to see whether any of these differences could be resolved by careful wording in POSIX. The LSB is also looking to see how many of these differences it can remove. Clearly, this also requires support of the upstream maintainers, the folks who write and support glibc, the Linux kernel, the various GNU utilities, etc. If code changes are required to achieve conformance to POSIX, how many existing applications will be broken? Developing standards can sometimes be a veritable tightrope walk between competing requirements.

Moving Toward Standardization

At the beginning of 2004, the LSB was almost ready to release version 2.0, a major overhaul of version 1.3. Version 2.0 contains several new libraries, notably, the standard C++ library. Many Independent Software Vendors (ISVs) had asked for C++ support in the LSB; it was one of the most common reasons people said that previous versions of the LSB were not useful.

One of the criteria for inclusion in the LSB is "best practice." You have to remember that standards are not typically at the leading edge of technology development. Good standards document existing practice, finding a common ground for multiple competing similar interfaces. It turns out that the C++ ABI is not yet particularly stable. GCC has had an unfortunate habit of changing the ABI for C++ with almost every release. When the LSB started the work of documenting the C++ ABI, GCC was at version 3.2. Midway through the LSB development, along came version 3.3, with a subtly altered ABI. Almost all the vendors said, "We are going to base our next product on GCC 3.3," so the LSB followed suit and upgraded the ABI specification to be based on GCC 3.3. Guess what: just as the LSB was about to release, GCC came out with 3.4, a new ABI.

To complicate matters further, the ISO (International Organization for Standards) had decided in 2003 that they wanted some standard "in the Linux space" to help keep up their relevance. The FSG was engaged in all of the discussions concerning this, and they proposed that the LSB was an appropriate answer.

Having the LSB become an International Standard like POSIX and ISO-C would help cement the position of Linux as an acceptable enterprise-level operating system. The ISO's Publicly Available Specification (PAS) process offers a way to transform standards developed by other groups into full-scale International Standards: first, the submitting organization is vetted and voted on by the members (countries), and then the document is submitted and goes through a six-month ballot to become an International Standard.

The FSG was accepted as a PAS submitter on October 31, 2003. They then had twelve months to submit a document. The plan was to submit 2.0 when it was published, which at that time was expected to be around February 2004.

Two factors then kicked in simultaneously: first, the C++ ABI instability issue, with distribution vendors arguing as to which version represented "best practice." The word "best" was originally supposed to mean "most widely used." In other words, if nine distributions were using GCC 3.3 and one was using GCC 3.4, then 3.3 was the "best." Of course, we all know there's another meaning to "best"—the motivation for the changes in the ABI from 3.3 to 3.4 was that 3.3 was buggy! The other thing that acted to delay an early release of LSB 2.0 was evaluating it in terms of "This is going to ISO: is it as good as it deserves to be?"

The answer to that last question landed directly on my own shoulders. I had to say to the LSB workgroup, "You know, this isn't a truly wonderful piece of work when you look at it in the light of an ISO standard." Thus started six months or more of hard slog to review and fix some of the inconsistencies, ambiguities, and other potential problems so that the FSG could submit it before 10/31/2004.

We made it on both fronts. Version 2.0 was published at the end of August, with the C++ material still based on GCC 3.3, but separated into a distinct module. That module was excluded from the material submitted to ISO but remained a required part of the FSG's certification program for LSB 2.0. To date, there have been 11 different products (distributions) certified against 2.0.

Since the C++ module was based on an outdated, buggy version of the GCC ABI, a roadmap was made for upgrading to GCC 3.4's ABI during the first quarter of 2005. It looks as though this version of the LSB (which will be called LSB 3.0, since the ABI itself has changed), will indeed have been published by the time you're reading this.

Meanwhile, the six-month ISO ballot is in progress. It started November 10, 2004, and finishes May 10, 2005. Votes here are by country. The U.S. has a vote, as do 21 other member countries (see http://www.jtc1.org under "Membership"). The procedure is somewhat complex: each member country is responsible for putting together its national body recommendation, but there are no overall rules as to how this recommendation is made. The U.S. does it one way, the U.K. another, and Australia yet another. On May 10, each national body sends its vote to the ISO secretariat in Zurich.

A country can vote yes, yes with comments, no (which must include substantive reasons), or abstain. The project editor has to address any no vote's comments, and doing so may change that no to a yes.

After the ballot has closed and the project editor has called and held a ballot resolution meeting to produce a "disposition of comments" report, the final votes are counted. If more than two-thirds of the votes are yes and fewer than one-quarter no, the document becomes a numbered International Standard. POSIX is ISO/IEC 9945, C is ISO/IEC 9899, and the LSB is destined to become ISO/IEC 23360.

As the LSB gathers momentum, we are starting to see public attacks on it from some organizations whose revenues could be directly affected by the widespread adoption of Linux at the enterprise level. This is proof positive that we are doing The Right Thing (TM).

ÆLEEN FRISCH

the bookworm

Æleen Frisch is a system administrator and writer living in Connecticut (www.aeleen.com).

aeleen@usenix.org

Books Reviewed in this Column

FORENSIC DISCOVERY

Dan Farmer and Wietse Venema

Addison-Wesley, 2004, 0-201-63497-X, 217 pp.

BUILDING A LOGGING INFRASTRUCTURE

Abe Singer and Tina Bird

Short Topics in System Administration 12, USENIX Association, 2004, 1-931971-25-0, 82 pp.

TROUBLESHOOTING LINUX FIREWALLS

Michael Shinn and Scott Shinn

Addison-Wesley, 2005, 0-321-22723-9, 381 pp.

INTERNET DENIAL OF SERVICE: ATTACK AND DEFENSE MECHANISMS

Jelena Mirkovic, Sven Dietrich, David Dittrich, and Peter Reiher

Prentice-Hall, 2004, 0-13-147573-8, 400 pp.

THE EXECUTIVE GUIDE TO INFORMATION SECURITY: THREATS, CHALLENGES, AND SOLUTIONS

Mark Egan with Tim Mather

Addison-Wesley Symantec Press, 2005, 0-32-130451-9, 288 pp.

SLAMMING SPAM: A GUIDE FOR SYSTEM ADMINISTRATORS

Robert Haskins and Dale Nielsen

Addison-Wesley, 2005, 0-13-146716-6, 420 pp.

SENDMAIL MILTERS: A GUIDE FOR FIGHTING SPAM

Brian Costales and Marcia Flynt

Addison-Wesley, 2005, 0-321-21333-5, 347 pp.

LINUX APPLICATION DEVELOPMENT, 2ND EDITION

Michael K. Johnson and Erik W. Troan

Addison-Wesley, 2005, 0-321-21914-7, 732 pp.

JAVA APPLICATION DEVELOPMENT ON LINUX

Carl Albing and Michael Schwarz

Prentice-Hall, 2005, 0-13-143697-X, 598 pp.

KNOPPIX HACKS: 100 INDUSTRIAL-STRENGTH TIPS & TOOLS

Kyle Rankin

O'Reilly, 2005, 0-596-00787-6, 314 pp. + CD.

I am thrilled and honored to be taking over from the illustrious Peter Salus as the Bookworm. This month's books are a somewhat random set, as the review copy pipes are just getting started for me.

FEATURED: *FORENSIC DISCOVERY*

Dan Farmer and Wietse Venema are best known collectively as the authors of the SATAN security evaluation application. However, they are also the authors of the one of the most important security articles in the literature, "Improving the Security of Your Site by Breaking into It," which explained the concept of (implicit) transitive trust. Venema and Farmer have collaborated on a new software tool, the Coroner's Toolkit, for performing post-break-in analysis; this is also the subject of their new book (the software appears at various points throughout the book and is discussed in an appendix). The book is divided into three parts, covering an overview of forensics and related concepts, fundamental system entities/data structures related to forensic discovery, and the specifics of data persistence.

This book is an excellent mix of the theoretical and the practical. Fundamental concepts are covered in detail, as are system data structures and entities (e.g., file systems, inodes, processes, system calls), providing a helpful—and necessary—knowledge base for the specific techniques for examining the latter that follow. As is typical of their work, Farmer and Venema have the knack of getting even experienced UNIX folks to look at and appreciate familiar things/material in a new way.

A significant thread running through the book is the persistence of data, including data deleted from memory and storage media. In the first chapter of the book's final part, the authors have compiled and performed a number of useful experiments that enable them to make very specific quantitative statements (e.g., how long trace data from a deleted file can be expected to linger under various system usage scenarios). This kind of information is needed to make both analysis and prevention planning practical (or even feasible).

This book is extremely useful for a wide spectrum of readers. People who are just getting started with computer forensic analysis will find it a clear and useful first book that will provide additional benefits on rereading. At the other extreme, experienced security administrators will appreciate the authors' clarity and insight in thinking about forensics in general and the specific discovery/analysis operations in particular. I always find that reading Dan and Wietse's work inspires me to strive for greater excellence in my own.

THE DEFINITIVE WORK ON LOGGING

Abe Singer and Tina Bird have done a marvelous job with the latest volume in SAGE's Short Topics in System Administration series. *Building a Logging Infrastructure* is a very comprehensive discussion of syslog and its uses and foibles. The work covers logging on both UNIX/Linux and Windows systems, and also discusses some replacements for syslog and when they are useful or necessary. The relative lack of material on log data reduction is due to the dearth of options in this area rather than the authors' omission. This book is so good that I find myself falling into a typical writer's emotional pitfall: being jealous that I didn't write it myself. A must-have for everyone.

THREE MORE ON SECURITY

The authors of *Troubleshooting Linux Firewalls* describe their books as including "the Tao of firewall security, the Zen of troubleshooting, and the nitty-gritty step-by-step instructions to fix a problem." This is a good description of their approach to their topic. The book does a very good job of handling the second and third of these items, which is the main thrust of their book. The first topic, firewall security, covered in two early chapters, seems a bit rushed (more space needs to be given to system hardening in particular). The book focuses on RedHat and SuSE Linux, but the principles and many of the specifics apply to almost any Linux distribution.

Internet Denial of Service: Attack and Defense Mechanisms is an in-depth treatment of DoS and DDoS attacks and ways of responding to and preventing them. The authors constitute a DoS dream team. Not surprisingly, the level of detail, understanding, and technical expertise is consistently high throughout the book. The book is also quite readable and even in tone and quality, despite having four authors. Don't let the book's goofy cartoon cover illustration mislead you or put you off. This is a serious book on an important topic.

Finally, *The Executive Guide to Information Security* is exactly what it sounds like: a book about computer security designed for nontechnical business executives. It places security concepts and practices within a typical business framework (mindset). As such, it may be useful to some readers of this column who need to present such technical information to managers and other nontechnical audiences.

SPAM IS THE WORD

Slamming Spam is another excellent book. It surveys all of the major spam-fighting techniques in common use today, covering both user-level and system-level strategies. It discusses—and provides detailed, correct directions for—configuring email clients ("user agents"), Sendmail, Postfix, qmail, Exchange, and Lotus Notes for spam filtering. It includes chapters on procmail, SpamAssassin, Vipul's Razor, blacklists, SMTP authentication, sender verification, and, of course, Bayesian filtering. Mail administrators, system administrators, and anyone who has to deal with a significant spam problem will find this book indispensable.

Sendmail Milters is the definitive reference for using Sendmail's filtering facility—milters—for dealing with spam messages. Its four parts cover the characteristics of spam, deploying a test environment for developing and testing, writing milters, and configuring Sendmail to use milters. This book is very comprehensive, extremely well written and eminently readable, and of the high quality one would expect from Brian Costales, working in concert here with Marcia Flynt.

A PAIR ON LINUX PROGRAMMING

Both of the programming books this time are aimed at beginners of various sorts. *Java Application Development on Linux* is written for readers with some programming experience and a basic familiarity with object oriented programming concepts. The first major section of the book introduces the Linux environment and the basic Java language, and the remaining four parts cover Java in action, including database queries, GUIs, Web interfaces, and distributed applications. The pace of the book is slow, but this is appropriate for its intended audience, and the book is long enough to introduce its topics in nontrivial detail. Students—and others—with only a moderate amount of programming experience will have no trouble working through this book on their own, and they'll be ready for advanced Java texts when they are through.

Linux Application Development is now in its second edition. Peter called the previous edition "a superb piece of work," while noting that it "makes no concessions to the unwashed." It would seem that much of the effort that went into the second edition was designed to broaden the book's appeal. For example, it now provides many, albeit brief, explanations of how Linux does things and also includes coverage of glibc. The book has been updated for the 2.6 Linux kernel and GNU C library 2.3. It continues to serve as an excellent reference for serious Linux application developers.

PARTING GLANCE

I'm going to try to end each column with a brief mention of something a little out of the mainstream. This time it's *Knoppix Hacks*, a book that made me nostalgic for the days when I frequently used a Yggdrasil install CD as a PC hardware diagnostic tool. Knoppix is a Linux distribution that runs directly from a CD without installation to disk. This book describes the many tasks it can perform, ranging from system configuration and repair to creating computer kiosks and roll-your-own Tivo-like systems. My one gripe with the book is its organization. Like all of the volumes in the O'Reilly Hacks series, it is structured as a numbered series of short articles, a scheme that inevitably favors breadth over depth. I'd say about two-thirds of the topics here are useful and the other one-third are cool, resulting in a book that is a good reference and also fun to flip open and start reading.

book reviews

RIK FARROW

rik@spirit.com

GOOGLE HACKS, 2ND EDITION

Tara Calishain and Rael Dornfest

O'Reilly, 2005, 0-596-00857-0, 443 pp.

When the first edition of *Google Hacks* came out, I ignored it. Sure, I thought I would learn something from the book, but Google seemed pretty easy to use as is. Then I heard Johnny Long (http://johnny.ihackstuff.com) talking about penetration testing using Google. Long opened my eyes to a lot of Google potential that I had been missing, and now I wanted to learn more.

This is not a review about Google Hacking for Penetration Testers. That book hasn't arrived yet (but I did ask for a copy). What showed up first was the second edition of *Google Hacks*, which seemed like a good place to start learning more about Google. And it is.

The first 28 pages cover the "basics" of Google, including useful special syntax such as site:, inurl:, and define:. Did you know you can include number ranges in your searches? Some of this information is so easy and useful, searching will never be the same.

But what are the other 415 pages for? The hacks in this book often require some programming, as hacks should. Unlike some recent O'Reilly books, this one is mostly for UNIX users, and it is useful to have not only Perl but also Python and PHP installed. If you want to try all the hacks, you will also need Java and .Net. And, yes, there are one or two hacks that will work only on Windows, and other things that work only if you have a Web server that can run scripts.

Why bother programming when you can just use the interfaces Google provides you? You can use Advanced Search or choose a date range for your search. Or you can instruct the Google search engine to include only pages indexed within a certain time period—but you must use Julian dates if you want to do so. A bit of Perl programming makes using Julian dates trivial to do.

Many hacks are pretty fanciful, such as a grid search, a popularity contest, Google mindshare, or finding a recipe to match the ingredients you have in your refrigerator. But there is lots of useful stuff, including an entire chapter about using the Google API, Gmail, how Google PageRank works, and adding Google searches to your own Web sites. If you are interested in improving the accuracy of your searches, or just want to have fun with Google, this book is a bargain.

KNOPPIX HACKS

Kyle Rankin

O'Reilly, 2005, 0-596-00787-6, 314 pp. + CD.

Ever wanted to turn someone on to Linux but shied away from having to take responsibility for supporting the installation? Instead of taking the plunge, just hand someone a copy of Knoppix on a CD (knoppix.net), and—as long as they have an i386-based system that can boot from the CD-ROM—they can experience Linux without installing it. Note that there are versions of Knoppix for some other architectures as well.

Knoppix Hacks is not for the casual explorer but for someone who wants to understand how to get the most out of Knoppix, because Knoppix is a lot more than a demo CD. I have been using Knoppix to teach my hands-on Linux security class for a year now. I customize my version of Knoppix by removing some packages and adding class exercises. When I first started doing this, I had to piece together the methods for working with Knoppix. This book provides details about creating your own custom Knoppix distribution in the final group of hacks. Wish I had had this a year ago.

Knoppix out-of-the-box is a fine toolbox. You can use it to replace lost passwords (including on Windows systems, with a downloaded utility), replace or fix boot loaders, repartition a system, and even recover a master boot block (as long as you don't use extended partitions). Knoppix can be a terminal server, DNS server, DHCP server, NFS server, Samba server, or Web server. Knoppix can also be used to rescue unbootable Windows systems through a registry edit or recovery of a CAB file.

Knoppix Hacks provides the information you need to make the most out of Knoppix. Without it, you would be hard pressed to discover all you can do with Knoppix. The book comes with an older version of a Knoppix CD (3.4), but even so, think of it as the perfect gift for a talented sysadmin, whether she works with UNIX or Windows.

FORENSIC DISCOVERY

Dan Farmer and Wietse Venema

Addison-Wesley, 2004, 0-201-63497-X, 217 pp.

Forensic Discovery is a book you must have if you are seriously interested in computer security. Farmer and Venema take you on a journey that covers just about anything that might remain in a computer as a result of an intrusion or other activity. Unlike other forensics books, the focus is not on finding evidence that can stand up in court. Instead, the authors

explore uncovering all the bits and pieces that might still be around months or years after an incident.

Farmer and Venema carefully lay the foundation for their methods of discovery. They explain booting, kernel initialization, system startup, file system details, process details, and examining malware in safety. They also dig deep into file systems, uncovering information about deleted files and information cached by journaling file systems. They offer thorough explanations that make it much easier to understand those normally ignored structures that underlie all modern file systems, yet are critical in forensics. Ever wonder just how long deleted data stays on UNIX systems? The authors explore persistence of data on disk and in memory through experiments, using real systems with different activity profiles to determine just how long data, or signs of intrusion, can remain in a system. The authors also discuss why uncovered data may only poorly represent the past, either because of normal system activity or active attempts at deception by miscreants.

While this book uses some of the tools developed as part of the Coroner's Toolkit, it is not a book about those tools. Rather, it is a serious exploration of how modern operating systems work in practice, what types of information get stored, how this information is stored, and techniques for retrieving and making sense of that data. The writing flows smoothly and clearly, with occasional geek humor, making this book easy to read and very accessible.

Even if you do not focus on security, you might want to read this little book just so you can have a better understanding of the systems you use and manage daily. The authors focus mainly on Solaris, Linux, and the BSDs. While Windows gets mentioned in passing, this is not a book for MSCEs. I highly recommend *Forensic Discovery* and am very glad it has finally been published.

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.

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*

Jon "maddog" Hall, *maddog@usenix.org*

Geoff Halprin, *geoff@usenix.org*

Marshall Kirk McKusick, *kirk@usenix.org*

EXECUTIVE DIRECTOR

Ellie Young, *ellie@usenix.org*

Years and Years Ago

Peter H. Salus
peter@usenix.org

2005!

Think about it!

Fifty years ago, 1955, IBM encouraged the first meeting of "users"—operators of the new 704—I guess, the first LISA. Also in 1955, AT&T/Western Electric submitted to the consent decree that, among other things, barred them from entering any business other than telephony or telegraphy (when was the last time you sent or received a telegram?).

A decade later, 1965. The Multics project has gotten underway. CTSS and DTS on the DEC-10 are the big thing. IBM is just coming out with the 360.

1975! Big time. The Labs have come out with V6. The "UNIX Users' Group" (linear parent of USENIX) has met (nearly two dozen in attendance!) and is bracing itself for another—publicized—meeting to be held 18 June. (Attendance bounded to 40.) It was the beginning of multiple meetings, too.

In October, Mel Ferentz chaired a meeting at CUNY, and Belton Allen chaired one four days later at the NPG in Monterey.

The year 1976 saw three meetings, too: one in Berkeley in February chaired by Bob Fabry; two at Harvard—April and October—both chaired by Lew Law.

The second of these was the first meeting to top 100 attendees, but the next May, Steve Holmgren chaired the first Midwest meeting at UIUC and eclipsed that with 250 in attendance.

In September, Oliver Whitby and John Bass ran a West Coast meeting at SRI with about 100 attendees.

The organization was about to become USENIX. The publication was about to become *;login:*.

An informal gathering of under two dozen people in May 1974 had turned into a semiannual event of major proportions.

[Blatant advertisement: Tom Limoncelli and I have edited an anthology of all the April Fool's Day RFCs. No Starch Press, out in July.]

Short Topics Booklets

Rik Farrow, Short Topics Editor
rik@spirit.com

The Short Topics in System Administration series of booklets are intended to fill a void in the current information structure, presenting topics in a thorough, refereed fashion but staying small and flexible enough to grow with the community.

Number 12 in the series, *Building a Logging Infrastructure,* by Abe Singer and Tina Bird, appeared first at LISA '04. As of March 1, a new booklet, *A Sysadmins' Guide to Oracle,* by Ben Rockwood, was beginning a technical review, and it should be in production early this summer. A contract has gone out to Xev Gittler and William Charles for a booklet with the working title *Being Root,* providing guidelines for working as the root user, focusing on best practices, tools, and ethics.

USENIX welcomes suggestions for topics, proposals for booklets, and technical reviewers. Interested? Please send email to sagebooklets@usenix.org.

For the list of booklets, see www.usenix.org/publications/.

Thirtieth Anniversary, USENIX Association

Peter H. Salus
peter@usenix.org

June 18, 1975. CUNY in Manhattan. Mel Ferentz runs the first USENIX conference. Of course, it wasn't called USENIX then, it was a UNIX users' group, until the lawyers at AT&T got tough about that (tm). And it wasn't the first meeting, either, as Lou Katz had run a small meeting in a conference room at Columbia in May 1974.

But there were "about 40 people from 20 institutions" at the 1975 meeting.

Look around at any USENIX conference, workshop, symposium. There'll be many times 40 folks. Yes, it has been 30 years, but the growth has come because USENIX has been where it's happening. And still is.

USENIX is where Kirk McKusick talked about memory management.

USENIX is where Tom Ferrin told us how to "cut this foil etch" and "insert this jumper wire."

USENIX is where we first heard about Tcl and OAK (= Java) and Perl and GNOME.

USENIX is where, in 1980 in Boulder, Colorado, Jim Ellis announced USENET.

USENIX is where UUNET began.

USENIX is where portability has been supported for 30 years.

USENIX has been sponsoring redistributable software since 1976.

USENIX held its first security workshop in 1988.

USENIX held a POSIX workshop in 1987.

And:

1st Graphics Workshop, 1985

1st C++ Workshop, 1987

1st Supercomputing Conference, 1987

1st Security Workshop, 1988

1st Mobile Computing Workshop, 1993

1st OSDI, 1996

1st Electronic Commerce Workshop, 1998

1st Embedded Systems Workshop, 1999

SAGE became a Special Technical Group of USENIX in 1992.

USENIX has brought together the core of the Linux Kernel development team in the Linux Kernel Developers Summit, held annually since 2001.

USENIX is where Ken Thompson spoke in 1974; where Steve Jobs spoke in 1987; where Stu Feldman lectured us on architecture; where we learned how Google works.

Oh, yeah. And how to fix your PDP-11 with this 98-cent resistor.

In 1966 BU (Before UNIX), Crispian St. Peters sang, "Follow me, I'm the Pied Piper . . ." It made it to #4 on *Billboard*'s list.

But if you follow USENIX, you'll really know where it's at.

conference reports

- Our thanks to the summarizers:

  Ashwin Bharambe
  Christopher Clark
  Priya Mahadevan
  Tipp Moseley
  Mohan Rajagopalan
  Marianne Shaw
  Alan Shieh
  Craig Soules
  Andrew Warfield
  Charles Weddle

OSDI '04: 6th Symposium on Operating Systems Design and Implementation

San Francisco, California
December 6–8, 2004

DEPENDABILITY AND RELIABILITY

Summarized by Christopher Clark

- ***Recovering Device Drivers***

  Michael M. Swift, Muthukaruppan Annamalai, Brian N. Bershad, and Henry M. Levy, University of Washington

  Awarded Best Paper!

  Michael Swift presented the first paper of the session on a technique for accommodating device driver failure.

  Building on the previous Nooks work on protecting the operating system when driver failures occur, the next challenge addressed is how to keep applications running when devices fail. This is achieved by introducing shadow drivers tasked with masking failures by acting as a "spare tire" in the event of an emergency.

  A single shadow driver is written for each device class, implementing the same interface to the operating system that a driver for a real device does. A shadow runs silently alongside each real driver, observing requests, until failure is detected by the OS. This triggers a restart of the real driver, while the shadow temporarily assumes responsibility for spoofing it and handling application requests until the restart completes. The shadow assists in reinitialization of the restarted driver by replaying previously observed configuration commands to the driver before handing back responsibility for requests.

  This scheme was implemented in the Linux 2.4.18 kernel for sound cards, network cards, and an IDE disk driver. Evaluation used both artificial fault injection of common programmer errors and deliberate porting of real bugs into the test kernel. Results showed that 98% of errors examined were recoverable using shadow drivers.

  George Dunlap (University of Michigan) inquired whether this work would reduce the inclination of companies producing drivers to bother removing bugs; Swift advised not to tell them we are doing this. Val Henson (IBM Research) offered praise and asked if there were plans to port the work into the mainstream Linux 2.6 kernel; Swift indicated not, given the current "grad student quality" of the code. A delegate from HP Labs likened the work to a dangerous condition for humans, where sufferers are unable to feel pain. Swift argued he would rather not experience the immediate consequences of driver faults, preferring instead to receive failure frequency statistics.

- ***Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines***

  Joshua LeVasseur, Volkmar Uhlig, Jan Stoess, and Stefan Götz, University of Karlsruhe, Germany

  Motivated by the desire to reuse existing device drivers written for commodity operating systems with the L4 research operating system, Joshua LeVasseur presented a method of using virtual machines to achieve this. The benefits are clear: they comprise a very large body of code and have undergone testing that would have to be repeated were they to be rewritten, if it were even possible to do so.

  The classic technique for driver reuse is to write "glue code," implementing the device driver API of the OS the driver was written for and performing translation

to the primitives of the new OS. The difficulty is that the driver API is often loosely defined, very wide, and messy, entailing manipulation of arbitrary data structures within the original OS, which consequently must be simulated by the glue code. The alternative approach proposed is to use the original OS itself as the glue code by running it atop a virtual machine monitor and exporting access to the device to other hosted virtual machines. Client VMs run a stub driver that communicates with the driver VM to achieve the desired use of a device. Memory protection hardware ensures that DMA initiated by device drivers is intercepted by the VMM and translated to indicate the correct physical addresses as necessary; with a paravirtualized VM, this step may be elided. A single machine arbitrates access to the PCI bus, with other machines communicating with this to implement higher-layered drivers.

A performance evaluation of the reuse of paravirtualized Linux drivers demonstrates single-digit percentage decrease for network and disk throughput, with a cost of approximately double the CPU consumption of the native driver.

Bin Ren (University of Cambridge) asked how the CPU scheduler selects which VM to run. LeVasseur responded that scheduling needed tuning to match the drivers in question. Val Henson noted that running multiple OSes to support the drivers increases the amount of code that needs to be correct for the system to function. LeVasseur countered that reuse of existing drivers increases confidence that the drivers are correct.

- ***Microreboot—A Technique for Cheap Recovery***

  George Candea, Shinichi Kawamoto, Yuichi Fujiki, Greg Friedman, and Armando Fox, Stanford University

  George Candea gave a talk on micro-rebooting, or restarting components within a system to purge any damaged state. The aim is to make reboot-based recovery fast in enterprise systems.

  Candea's thesis is that improved availability can be achieved if it is possible to restart only the faulty parts of the system. This is an argument for making component restart possible and fast; to enable this, one must refactor code to move session state in a separate lump, distinct from the ephemeral state within the component. The session state alone may then persist between component restarts, and the micro-reboot can address the symptoms of software failure, such as transient exceptions, deadlocks, and memory leaks.

  To evaluate the approach, a prototype auction Web service was constructed from J2EE components running on the JBoss platform. Since rebooting individual components is significantly faster than rebooting the entire JVM and has little discernible impact on availability, the fault detector need not be very accurate, because false positives are not expensive. This encourages a strategy of micro-rebooting aggressively, or periodic "micro-rejuvenation" to keep the system healthy.

  A delegate from Georgia Tech noted that in real applications there may be significant dependencies between components, and so the restarting of a single component may require the restarting of almost the entire system. Candea replied that the common design patterns encouraged by the use of EJBs make such tight coupling infrequent. A delegate from Rice University asked about the complexity of the prototype and whether it was entirely in Java or included modifications to the runtime. Candea responded that the JVM was not modified, 200 lines of code were added to JBoss, and the remaining implementation was composed of managed EJBs.

AUTOMATED MANAGEMENT I

Summarized by Andrew Warfield

- ***Automated Worm Fingerprinting***

  Sumeet Singh, Christian Estan, George Varghese, and Stephan Savage, University of California, San Diego

  This work addresses the problem of identifying worm outbreaks as quickly as possible. Sumeet began by identifying the design space and clarifying the three main requirements of their approach: response time, granularity of containment, and deployment. Their system, dubbed "EarlyBird," aims for a response time on the order of seconds, contains worms at a precise content signature granularity, and is network (rather than end-host) based. The key challenges in developing such a system are processing and storage: at gigabit line rates, packets must be processed in 12 microseconds or less, and log data accumulates very quickly.

  EarlyBird capitalizes on two properties of worms for fast identification: content prevalence, which involves the frequency of the substring, assuming that there is an invariant substring across all instances of a particular worm, and address dispersion, the property that such substrings will travel between a large set of hosts. Their approach uses fixed-length substring hashes to find common signatures. Hashing has the additional benefit of allowing value sampling, through only examining packets in a specified range of hashes. EarlyBird has been deployed for eight months in separate academic and ISP environments, and has found all known worms as well as several new ones.

  Petros Maniatis from Intel Research asked about how EarlyBird would fare against polymorphic worms. Sumeet answered that most encrypted worms were easy to classify because embedded decryptor code is invariant. Concerning SSH tunneling, he proposed that encryption be made

gateway-to-gateway rather than end-to-end. Regarding harder polymorphism, for instance NOOP insertion, he said that more investigation was required. Brad Karp (Intel Research) asked how many hosts would be exploited before prevalence thresholds were crossed, and mentioned the 30/30 rule (the address dispersion threshold of 30 sources and 30 destinations) described in the paper. Sumeet agreed that this issue needed more consideration but said that they had moved on to better methods than 30/30 since the paper was written. Someone from Microsoft Research asked whether worms could circumvent EarlyBird's detection mechanisms. Sumeet explained that value sampling was difficult to outguess. Someone from Rice University asked about worms over P2P. Sumeet answered that in general these were not a problem, but that in some specific cases, such as BitTorrent, there is a risk of generating false positives.

- ***Understanding and Dealing with Operator Mistakes in Internet Services***

  Kiran Nagaraja, Fabio Oliveira, Ricardo Bianchini, Richard P. Martin, and Thu D. Nguen, Rutgers University

  Operator-caused outages are a major problem in Internet services. The authors explored ways to provide "realistic virtual environment"-based support to help prevent such mistakes. The talk was divided into three parts: understanding operator mistakes, dealing with the problem, and validation of their results.

  In order to better understand operator error, the authors conducted a study involving 43 experiments and 21 operators of varying levels of experience. Operators were asked to perform a variety of both proactive and reactive tasks on a model three-tier Web service environment, and the experiments resulted in a total of 42 operator mistakes. The highest categories of mistakes were those resulting in degraded throughput or inaccessible service, most frequently as a result of configuration problems. In order to prevent operator mistakes from happening, the authors developed a virtual environment in which system changes could be tested before they were applied. Their system could be run online, by "shunting" the request stream, or offline using traces. Finally, Kiran presented a validation of their prototype. In a set of live operator experiments using their environment, six of nine mistakes were caught before being applied to the production system. The authors also emulated the operators from the initial experiments and were able to catch 28 of the 42 mistakes that were observed there. The most frequently corrected mistakes were those of global configuration and starting the wrong version of a service.

  Andrew Whitaker (University of Washington) asked if mistakes had been symptomatic of operator unfamiliarity with the model three-tiered service. Kiran acknowledged that this was worth considering in future work but that this was the best initial approach to the study. Jonathan Appavu (IBM Research) asked what other tools might be developed to help operators avoid mistakes. Kiran pointed out that the biggest problem is that many tools, such as configuration checkers, are very application-specific. While these tools are useful, he pointed out that the approach described in the paper worked with all applications and tested the actual results of operator actions.

- ***Configuration Debugging as Search: Finding the Needle in the Haystack***

  Andrew Whitaker, Richard S. Cox, and Steven D. Gribble, University of Washington

  Andrew began by describing his work as a different sort of debugger, one targeted at configuration errors. As an initial example he presented the issue of Mozilla crashing sometime after a set of extensions had been installed and observed that current approaches to the problem might involve googling for "mozilla crash" or reading help menus. Unfortunately, in many cases these will not provide a solution, and even reinstalling the broken application will not fix the problem. Their work aims to provide tool support to systematically identify the causes of this class of "worked yesterday, not today" (WYNOT) configuration errors.

  Chronus is a tool that the authors have developed to isolate WYNOT errors. In Chronus, the operating system runs in a lightweight virtual machine above Denali, a virtual machine monitor that the group developed previously. Denali allows block devices used by the operating system to be made into "time travel" disks, logging a history of all past states of the disk. To diagnose a configuration error, the user provides a probe script that tests for the existence of the error. Chronus will then do a binary search across a specified region in the history of the disk, booting the OS and running the probe, finally returning a pair of disk images in which the probe results transition from success to failure. By examining the differences between these images, Chronus will provide the user with the specific block update that resulted in configuration error. Returning to his Mozilla example, Andrew observed that the search process finished while he got coffee, and it isolated the error to the specific extension that was crashing Mozilla.

  Chi Zhang (Princeton University) asked whether Chronus would be able to identify problems that resulted from large logical disk transactions, for instance the installation of new software packages. Andrew answered that extra tools would be required to identify larger-granularity causes. Shinji

Suzuki (University of Tokyo) asked a follow-up question regarding the effects of buffer caches, which Andrew agreed were also a problem. A third question regarded issues such as spyware, in which failure might occur after a disk is changed. Andrew pointed out that this was discussed in more detail in the paper, but that spyware was a problem in general. Finally, David Oppenheimer (University of California, Berkeley) asked Andrew to comment on the difficulty of probe writing, especially for difficult-to-test issues—for instance, changing Mozilla's text from English to Japanese. Andrew pointed out that Chronus's contribution was to change a very time-consuming class of configuration debugging problems into software testing problems. He agreed that while probes could be difficult to write, he felt that they could be developed by experts and reused in many situations.

FILE AND STORAGE SYSTEMS I

Summarized by Craig Soules

- ***Chain Replication for Supporting High Throughput and Availability***

  Robbert van Renesse and Fred B. Schneider, Cornell University

  This talk, given by Robbert van Renesse, described chain replication, a system for high-throughput replication. Their system services two types of requests: updates and queries. The storage nodes are formed into a chain, and then updates are sent to the head of the chain and queries are sent to the tail. When a storage node sees an update it processes and then forwards it to the next server in the chain. Once the request has been processed by the tail, it is guaranteed to have executed on all of the storage nodes, and an acknowledgment is sent to the client. This same guarantee means that any queries processed at the tail will return data seen by all storage nodes.

  In this system, a master maintains the chain membership. Node failure is handled by the master and the previous node in the chain coordinating to remove the failed node. Nodes may only be added to the tail of the chain, and their content must first must be synchronized with the existing tail. Once synchronized, the master and the tail coordinate to move the node into the tail. All new queries are then sent to the new node. Any lost requests must be resubmitted by clients (consistency is maintained via the requirement that all updates be idempotent).

  The results of this work indicate that chain replication can provide higher throughput than most primary/backup systems. David Shultz (MIT) asked how the system handled network partitions to coordinate chain formation. He was told that the configuration of chains among masters is done using Paxos, which automatically handles such failures. Jay Lorch (Microsoft Research) asked if they had examined weak-consistency chain replication. He was told that weak-consistency chain replication performs identically to weak-consistency primary/backup.

- ***Boxwood: Abstractions as the Foundation for Storage Infrastructure***

  John MacCormick, Nick Murphy, Marc Najork, Chandramohan A. Thekkath, and Lidong Zhou, Microsoft Research Silicon Valley

  This talk, given by Lidong Zhou, described Boxwood, a toolkit for developing distributed storage abstractions. Today, distributed storage systems make use of reads and writes of blocks (or objects) and strict interface primitives that sometimes make it difficult to layer more complex abstractions (e.g., b-trees, hash tables) on top of them while still maintaining the same guarantees of consistency and scalability.

  Boxwood provides several tools needed to create more complex data abstractions: a lock service, a logging service, a consensus service, and a replicated chunk store. The first three of these can be used by the application to provide consistency to the new algorithm. The chunk store provides a replicated virtualized address space to store data across a large set of machines, thus providing reliability and scalability of data storage.

  Lidong then described an example distributed algorithm they built by taking an existing non-distributed b-link tree algorithm and hooking it into the Boxwood abstractions to provide a distributed version of the algorithm. With this new distributed b-link tree in place, they were then able to layer an entire distributed file system, labeled BoxFS, over the abstraction. Performance analysis of BoxFS indicates excellent scaling from two to eight servers, and better performance than a system running NFS over NTFS.

  Erik Reidel (Seagate) asked about their performance graph that indicated they were getting 0.5 MB/s throughput using 5 to 40 disks. Lidong answered that the graph was supposed to show the effect of lock contention on concurrency rather than actual throughput. Erik then asked how many clients were used in these experiments. Lidong responded that they used two to eight mount points with one client per mount point.

- ***Secure Untrusted Data Repository (SUNDR)***

  Jinyuan Li, Maxwell Krohn, David Mazières, and Dennis Shasha, New York University

  This talk, given by Jinyuan Li, described SUNDR, a system for ensuring tamper detection of files stored on untrusted data repositories. They began by describing a new kind of consistency: "fork consistency." Fork consistency guarantees that if a server provides two clients with different copies of the same file, that it can never again provide those clients with the same copies of that file. Also,

the server is unable to tell the client that their change has not been applied once it has agreed to the change.

To achieve this, SUNDR stores a version vector with each file. This vector contains a version number for each client of the file. When a client makes a change to the file, it obtains the file and the file's version vector. The client can then compare the provided version vector with its current vector from its last access of the file. If its stored vector is not an exact subset of the provided vector (i.e., its version numbers are all less than or equal to the provided ones, and the version number for that client is identical), then it can detect that the server has broken fork consistency. Undetected modification of the version vector is prevented using signatures.

Emin Gun Sirer (Cornell University) asked about the possibility of a malicious client using replay attacks to make the server appear faulty. Jinyuan indicated that even a colluding server and client could not break the fork consistency. Algis Rudys (Rice University) asked at what data abstraction level this would be most useful. Should it be coupled with each NFS operation? Each block? Jinyuan responded that SUNDR currently works at the block level, but that the techniques could be applied to any of these levels.

DISTRIBUTED SYSTEMS

Summarized by Priya Mahadevan

- ***MapReduce: Simplified Data Processing on Large Clusters***

  Jeffrey Dean and Sanjay Ghemawat, Google, Inc.

  MapReduce is a programming model with an associated implementation for processing extremely large input data sets. Along with the programming model, MapReduce also automatically handles fault tolerance, I/O scheduling, load balancing of input data set among various machines, and inter-machine communication.

  The programming model is designed such that the input and output is a set of key/value pairs; the computation is expressed as two functions, map and reduce, both of which are user specified. On supplying the input key/value pair, the map function produces a set of intermediate key/value pairs. The intermediate key/value sets are then used by the reduce function to produce the output. An example of this type of programming model is counting the occurrences of a specific word in an input data file or files.

  There are two kinds of programs, master and worker. The master program is special, in that it delegates tasks to the workers. Map Reduce also handles the following functionality:

  - Parallel execution: Input data is split into tasks, and each task is executed on different sets of machines. Users can also specify a partitioning function for this purpose.
  - Fault tolerance: Failures are detected using periodic heartbeats, and in-progress tasks are then executed on other machines.
  - Dynamic load balancing: The master takes proximity of the workers into consideration (with respect to location of the input data) while assigning tasks to the workers.

  In addition several refinements—skipping bad records, generating sorted output files, providing status pages that indicate tasks in progress, etc.—are provided by MapReduce. The performance was tested for two benchmarks, grep and sort, on a cluster comprising 1800 machines. In conclusion, MapReduce simplifies large-scale computations, and since it handles most of the parallelization and distributed systems internals, users without experience in parallel and distributed systems can use it effectively.

  A member of the audience wanted to know of any task that could not be handled using MapReduce. The answer was join operations could not be performed with the current model. Someone else wondered how MapReduce differs from parallel databases. MapReduce data is stored across a large number of machines as compared to parallel databases, the abstractions are fairly simple to use in MapReduce, and MapReduce also benefits greatly from locality optimizations.

- ***FUSE: Lightweight Guaranteed Distributed Failure Notification***

  John Dunagan, Michael B. Jones, Marvin Theimer and Alec Wolman, Microsoft Research; Nicholas J. A. Harvey, Massachusetts Institute of Technology; Dejan Kostić, Duke University

  Managing failures in a distributed application is a challenging task: one needs to maintain a lot of state, and handling cascading failures require handling many different cases. FUSE is a failure notification mechanism that addresses the above issues. FUSE is not a failure detection service; it requires the participation of applications to guarantee failure notification. Examples of applications that could benefit from FUSE include peer-to-peer storage, multicast trees, and content distribution networks. The advantages of using FUSE include guaranteed failure notification, convenient handling of all corner failure cases, and reduction in distributed application complexity.

  Applications create a FUSE group by specifying the participating nodes, and FUSE guarantees that every member in this group will be notified whenever a failure condition affects this group. By creating a spanning tree among the group members, FUSE can guarantee failure notification; it does not

need to monitor all the paths between all the nodes.

FUSE can tolerate arbitrary network failures and node crashes, but it cannot handle byzantine failures. Applications need to handle such failures explicitly. The FUSE API comprises three methods: CreateGroup, RegisterFailureHandler, and SignalFailure. The authors implemented FUSE over the SkipNet overlay, so that they could take advantage of the DHT's liveness checking properties. Using a DHT also assures low network costs even when there are many groups.

FUSE was evaluated on the ModelNet testbed. Evaluation metrics included group creation latency time, failure notification latency, performance under churn, and false positive rates. During Q&A, someone asked whether FUSE could pinpoint which node in the group failed rather than simply notifying group members about a failure. It turns out that FUSE cannot notify the exact node where the failure occurred. Someone was concerned about how FUSE could handle transient network failures such as a certain node failing and recovering before all the group members could be notified of the news of the failure. The speaker said there is no good way to handle such a situation.

- ***PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-Area Services***

  Ming Zhang, Chi Zhang, Vivek Pai, Larry Peterson, and Randy Wang, Princeton University

  Anomalies in Internet routing are common, and detecting them is a nontrivial task. The irregularities could reside in either the forward or reverse paths and are hard to isolate. The contributions of this paper include large-scale study and classification of routing anomalies and techniques for anomaly detection and isolation.

  PlanetSeer combines passive monitoring with active probing. Probes are sent only during the period of anomaly, so there is low network overhead. Both modules use TTL change and *n* consecutive timeouts in TCP flows for the detection mechanism. The probing module is made up of baseline probes (when a new IP appears), forward probes (when a possible anomaly is detected), and reprobes (to find duration of an anomaly). By clustering nodes based on their geographic location and choosing a node in each group for probing purposes, the probing overhead is reduced. The authors also use traceroute from multiple vantage points to narrow down anomaly location.

  Some of their results:

  - The authors found approximately two anomalies per minute over a period of three months.
  - Tier-1 autonomous systems (ASes) account for the least number of persistent and temporary loops, path changes, and outages, while tier-3 ASes account for the largest.
  - Temporary loops have much longer hop lengths than persistent loops. Persistent loops either get resolved very quickly or stay for a very long period of time (> 7 hours).
  - Outages occur closer to network edge, while path changes have a much wider impact.

  One of the more interesting questions posed was whether any correlation was observed between the anomalies observed at the rate of two per minute. The speaker replied that while they did not explicitly look at anomalies correlation, his guess was that they were correlated.

NETWORK ARCHITECTURE

Summarized by Ashwin Bharambe

- ***Improving the Reliability of Internet Paths with One-Hop Source Routing***

  Krishna P. Gummadi, Harsha V. Madhyastha, Steven D. Gribble, Henry M. Levy, and David Wetherall, University of Washington

  Krishna Gummadi began by stating that recently proposed overlay designs (RON, Detour) for improving Internet path reliability were overly complex. This observation was supported by a detailed study using a Planetlab testbed to measure Internet path-failures. About 3000 different types of destinations (including commercial servers and broadband home users) were probed from Planetlab nodes. A failure was defined as three consecutive TCP RST losses in response to TCP ACKs combined with a traceroute failure to the destination. The observed path-failure rates were four per week for servers and seven per week for broadband hosts. Most paths witnessed at least one failure every week. Furthermore, last-hop failures for servers were infrequent, implying that unavailability of servers could very well be due to path failures in the network. The conclusion is that while failures definitely exist, they are uncommon and short, and mechanisms to overcome these should themselves be lightweight.

  Gummadi went on to propose a new scheme called Simple One-hop Source Routing (SOSR), which achieves the above objectives. The idea is simple: Instead of using complex multiple-hop overlay routes, end nodes just utilize one intermediary node for "routing around" in case of a failure. Several measurements were performed to understand the utility of such an approach. It was found that in most cases, the failure could be avoided by using any one of a large set of intermediaries.

The authors showed that the random-4 strategy (picking four random intermediaries) provides most of the possible benefits. Notice that this scheme does not require any a priori probing either by end nodes or by the intermediaries, and hence is stateless. Gummadi concluded by stating that in spite of these positive results, it is unclear whether end users will be able to perceive performance improvements due to SOSR, since multiple orthogonal factors contribute to overall end-user perception.

- ***CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups***

  KyoungSoo Park, Vivek S. Pai, Larry Peterson, and Zhe Wang, Princeton University

  Most of the previous studies of the DNS infrastructure have ignored the impact of local DNS name servers (LDNS) on performance. In this talk, KyoungSoo Park, using a comprehensive set of measurements, showed that client-side DNS (LDNS) failures are widespread and frequent and reduce overall performance and availability. LDNS servers belonging to several Planetlab sites (these servers are site-specific and not tied to Planetlab) were monitored for an extended period by issuing trivial local name lookups. While most of the lookups took minimal time to complete (as expected), a surprisingly heavy tail was observed for the lookup times. Furthermore, such delays were widespread, across several sites, and were frequent. The authors cited two principal causes for this effect: overloading of local nameservers due to heavy memory pressure, and lack of maintenance.

  The authors propose an incrementally deployable cooperative DNS lookup scheme (CoDNS) with an aggressive adaptive timeout to overcome LDNS problems. The basic idea is to forward a name lookup to one or more DNS servers at other sites when LDNS is suspected of failing. The choice of which servers to contact is determined by their proximity to the querying server as well as availability. The authors showed that CoDNS is able to remove the heavy tail of lookup times and add an extra "9" to the availability of the local DNS infrastructure.

  A few issues were raised during the Q&A session: Andrew Myers (Cornell University) worried that CoDNS reduced the security of an already insecure and critical infrastructure. David Oppenheimer stated that a simple tweak to existing DNS implementations (viz., adding *remote* secondary nam servers in configuration files, and reducing the default timeout values) would essentially provide the benefits of CoDNS.

- ***Middleboxes No Longer Considered Harmful***

  Michael Walfish, Jeremy Stribling, Maxwell Krohn, Hari Balakrishnan, and Robert Morris, MIT Computer Science and Artificial Intelligence Laboratory; Scott Shenker, University of California, Berkeley, and ICSI

  Middleboxes are defined as entities interposed between end hosts (over the Internet) that perform more tasks than plain IP forwarding. Several such middleboxes (e.g., NATs, firewalls, caches) are in common use. However, because they violate the end-to-end principle, middleboxes are not in harmony with the existing Internet architecture, despite their clear practical benefit. In this talk, Michael Walfish presented an architectural extension to the Internet (Delegation-Oriented Architecture, or DOA) to accommodate such middleboxes.

  DOA is composed of two fundamental primitives: First, each endpoint (middleboxes are also considered endpoints) has a globally unique topology-independent identifier called an EID; second, receivers and/or senders can *invoke one or more endpoints as delegates* for routing messages. By treating NATs and firewalls as such delegates, DOA can elegantly incorporate middleboxes into the overall design. Furthermore, the DOA permits the existence of new functionality, such as off-path firewalls where entities external to an organization can offer firewall services.

  In order to implement these primitives, an infrastructure for resolving flat EIDs to physically routable identifiers (IP addresses) is essential. The authors present a global DHT as a promising candidate for such a resolution infrastructure. While security and performance of the resolution infrastructure have been addressed to some extent by the authors in the paper, Steve Gribble (University of Washington) argued in the Q&A session that the next critical challenge for DOA is investigating suitable mechanisms for maintenance and troubleshooting.

AUTOMATED MANAGEMENT II

Summarized by Marianne Shaw

- ***Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control***

  Ira Cohen, Moises Goldszmidt, Terence Kelly, and Julie Symons, Hewlett-Packard Laboratories; Jeffrey S. Chase, Duke University

  Ira Cohen presented an approach for automatically inducing models of system performance; the technique requires little or no domain-specific knowledge, and therefore can be applied to a wide variety of systems.

  The motivation behind this work is that we, as a community, have figured out how to build complex, large-scale network services; we've instrumented those services to capture a large number of diverse performance metrics. However, for any particular failure or event, how do we know which metric or set of metrics we should be looking at? Which metrics will not

help in determining the root cause of the problem?

This work automates the analysis of this large collection of instrumentation data using Tree-Augmented Naïve Bayesian networks, or TANs. By capturing traces of both normal and anomalous events from an instrumented three-tier Web server, and combining that with instances of Service Level Objectives (SLO) failures, TANs are used to produce performance models. These models can be used to select the set of gathered metrics that correlates strongly with higher-level Web server performance.

In evaluating their approach, several key observations were made. Small sets of metrics are much better than a single metric at predicting system behavior; for the Web server workload, it was typically three to eight metrics. Because each metric in the set is associated with a particular system component, the set can provide assistance identifying the root cause of anomalous behavior.

Questions focused on how to use the approach. If you are interested in the dynamics of the system rather than a binary observation, could you still use this technique? Yes, if you could convert those dynamics into a binary classification. Is it possible to use this technique for prediction of input? The authors do not yet have sufficient experience to know what the generated models will look like.

- ***Automatic Misconfiguration Troubleshooting with PeerPressure***

  Helen J. Wang, John C. Platt, Yu Chen, Ruyun Zhang, and Yi-Min Wang, Microsoft Research

  Helen Wang presented PeerPressure, a mechanism for troubleshooting misconfigurations in modern, complex operating systems and applications. PeerPressure uses Bayesian statistics to compare the Windows registry of a misconfigured machine with a collection of Windows registries from other machines; the statistics can then be used to find the misconfiguration and fix it.

  PeerPressure embraces the conformity of computer systems and their configurations, and the belief that most applications work correctly on most machines. When an application is deemed to be working incorrectly, its associated Windows registry entries ("suspects") are captured by the user and fed into PeerPressure. Suspects are canonicalized and statistically compared with the collection of sample Window registries to generate a ranking based on the probability that a suspect is misconfigured. PeerPressure uses this ranking to modify entries in the Windows registry one by one until the configuration problem is resolved.

  Twenty real-world "troubleshooting" problems and a database of 87 machines' Windows registries were used to evaluate PeerPressure. The system was able to diagnose the misconfiguration problem in 12 of these 20 cases, and to significantly narrow down the set of possible misconfigured entries for the remaining eight.

  To demonstrate the obscurity of various misconfigurations, Helen introduced and showed the consequences of a misconfiguration error in the Windows registry during her talk.

- ***Using Magpie for Request Extraction and Workload Modeling***

  Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier, Microsoft Research, Cambridge, U.K.

  Rebecca Isaacs presented the use of the Magpie toolchain for automatically generating models of a system's workload that can be used for performance debugging, anomaly detection, and capacity planning.

  Magpie is designed as an online mechanism, so it must handle intermingled requests, unrelated operating system and application events, cross-machine interactions, and monitoring of resource consumption in a lightweight, non-obtrusive manner. Therefore, rather than tagging each request flowing through the system, Magpie uses an application-specific schema to correlate system events corresponding to the same request. A request parser uses this schema while processing an event log to correlate events. These events are then associated with a particular request using a technique called "temporal joins," which attributes events to the same request if they could have occurred during the same valid interval.

  Magpie was validated against traces of synthetic workloads and shown to be feasible for a two-tier Web server and the TPC-C Benchmark Kit. Someone asked whether they had looked into its applicability to real-world systems yet. While they would like to look at large distributed systems, currently they have only looked at two- or three-machine systems; they need to scale out to larger systems. Magpie does require application instrumentation, so it will require effort to apply to existing systems; they have been evangelizing to try to get instrumentation added to products.

BUGS

Summarized by Mohan Rajagopalan

- ***Using Model Checking to Find Serious File System Errors***

  Junfeng Yang, Paul Twohey, and Dawson Engler, Stanford University; Madanlal Musuvathi, Microsoft Research

  Awarded Best Paper!

  This paper, presented by Junfeng Yang, was about identifying file system bugs by using model checking techniques. These bugs are potentially destructive, but traditional testing techniques have

been ineffective due to the exponential possibilities that one needs to consider. The talk described a file system model checker called FiSC, based on the CMC framework, which was effective in finding bugs that would otherwise have been very difficult to detect using static analysis techniques.

Of the several interesting components that make up the system, the talk dealt primarily with state reduction for the model checker. The checking process starts with some state and sees whether the state was encountered previously. Instead of a randomized approach for a state-space search, they advocate a guided search for their testing. Consistency checks are performed through an abstract file system that models the file system (e.g., for tracking topology), and this can be compared with the model to check for errors. To handle journaling file systems they resort to logging. This description concluded with the observation that checking could be made more thorough by downscaling and via canonicalization.

George Candea (Stanford University) asked about the modifications required to apply this system to identify bugs in databases. Junfeng noted that this may be easy to incorporate.

- ***CP-Miner: A Tool for Finding Copy-Paste and Related Bugs in Operating System Code***

  Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou, University of Illinois, Urbana-Champaign

  Zhenmin Li described a technique to identify "copy-paste" bugs in operating systems by adopting a programmer's perspective rather than software analysis. Zhenmin noted that in principle this work was similar to, and was in fact motivated by, plagiarism detection tools such as MOSS and JPlag. While the software engineering community has taken a recent interest in identifying copy-pasted code, existing tools have several shortcomings, such as high cost, inaccuracies, etc.

  The basic idea is to apply subsequence matching to identify code that has appeared at least twice, an idea frequently used in data mining. The algorithm is based on identifying frequent sequences, building a sequence database, and composing (joining) sequences within the database. This process is repeated several times. The talk also described an example where their technique was able to identify a "forget to change" bug—where the programmer forgets to replace variable names in a copy-pasted segment of code.

  The first question was whether their tool was suitable for other large systems and if they had tried it out elsewhere. Zhenmin replied that while they had only tried it on small software benchmarks, it could be suitable. Another interesting remark was that comments can be very useful in identifying copy-pasted code. Finally, someone asked whether their system could mine CVS code repositories. Zhenmin replied that this was something they were currently looking at.

- ***Enhancing Server Availability and Security Through Failure-Oblivious Computing***

  Martin Rinard, Cristian Cadar, Daniel Dumitran, Daniel M. Roy, Tudor Leu, and William S. Beebee, Jr., Massachusetts Institute of Technology

  Martin Rinard presented a very interesting and entertaining paper on a controversial new concept, "failure-oblivious computing," which differs from the traditional fail-stop philosophy used to build computer systems. The driving principle here is that programs are complex and should be able to tolerate localized memory errors. The talk began with a discussion of bounds violations in the standard C model. An empirical evaluation of five "failure-oblivious" programs was then presented by comparing these programs to their regular counterparts in the context of security, initialization, correct continuation, and ability to handle attack input. While failure-oblivious programs had some limitations, the results of this evaluation looked promising.

  During Q&A, someone asked whether this meant that bugs should not be fixed and we should not bother about them. Martin's reply was that with failure-oblivious computing they are no longer bugs, so the program should do what it's doing. Zin Dong (Princeton University) pointed out that while this idea would be useful for some things, it may not be able to handle linked structures. Rob Pike (Google) mentioned that he did something similar with data mining, and it would be more comforting to know that all the failures were stored in a log. Margo Seltzer (Harvard University) asked Martin to compare this to sandboxing systems; Martin replied that this was much simpler. Dawson Engler (Stanford University) noted that it may not be possible to track race conditions this way. Another interesting question was what if an attacker knew that the application being targeted was failure oblivious. Someone pointed out that this approach could be very frustrating, especially in pinpointing bugs, since a program would continue even when you want it to fail. Jay Lepreau from Utah mentioned that this would be analogous to testing when optimizations are turned on.

WORK-IN-PROGRESS REPORTS

Summarized by Tipp Moseley

- ***pDNS: Parallelizing DNS Lookups to Improve Performance***

  Ben Leong and Barbara Liskov, MIT

  Up to 10% of DNS queries exceed 2s of latency. To hide this latency, overlay networks of resolving nameservers are cached and queried in parallel. This results in a latency being the maximum

latency of *N* queries instead of the sum of the latencies of *N* queries.

- ***Trickles: A Stateless Transport Protocol***

  Alan Shieh, Andrew Myers, Emin Gun Sirer, Cornell University

  Typical protocol stacks require resources, limit scalability, are vulnerable to DoS, and are barriers to migration. Trickles proposes to move all state to the client, via continuations, which are self-describing and encapsulate server state. This enables transparent failover, load balancing, and anycast services.

- ***Surviving Internet Catastrophes***

  Flavio Junqueira, Ranjita Bhagwan, Alejandro Hevia, Keith Marzullo, and Geoffery M. Voelker, University of California, San Diego

  In order to improve server uptime and protect important data from attacks from worms, data must be duplicated across different operating systems and configurations. This approach will differentiate exploitable flaws in software, and is successful in surviving past and even more aggressive worms at low cost.

- ***Honeycomb: Enabling Structured DHTs to Support High-Performance Applications***

  Venugopalan Ramasubramanian, Yee Jiun Song, and Emin Gun Sirer, Cornell University

  DHTs show great promise to run infrastructure services because they are self-organizing, failure resilient, and highly scalable. Honeycomb investigates the space-time tradeoff in caching data, and guarantees <1 hop average lookup performance while minimizing resource consumption.

- ***PRACTI Replication for Large-Scale Systems***

  Mike Dahlin, Lei Gao, Amol Nayate, Arun Venkataramani, Praveen Yalagandula, Jiandan Zheng, University of Texas at Austin

  - PRACTI focuses on several principles involving replication for large-scale systems: separate mechanism from policy, and separate data and control paths. This result is a universal replication toolkit with the following attributes:
  - Partial replication: an order-of-magnitude less bandwidth and storage space
  - Topology independence: reduced time taken to synchronize
  - Arbitrary consistency: improved availability in disconnected operation

- ***Shruti: Dynamic Adaptation of Aggregation Aggressiveness***

  Praveen Yalagandula, Mike Dahlin, University of Texas at Austin

  Shruti is a dynamically adapting, lease-based mechanism that adapts based on read/write history.

- ***MOAT: A Multi-Object Assignment Toolkit***

  Haifeng Yu, Phillip B. Gibbons, Intel Research Pittsburgh

  Heavy user accesses to shared files requires replication of data objects and files. The goal of such a system is high availability for multi-object accesses, and the key issue of the problem is replica assignment. MOAT is the first system to observe the importance of replica assignment, shows strong theoretical results regarding best/worst assignments, and implements a toolkit for replica assignments.

- ***Causeway: Operating Systems Support for Distributed Resource Management, Performance Analysis, and Security***

  Anupam Chanda, Khaled Elmeleegy, Nathan Froyd, Alan L. Cox, John Mellor-Crummey, Rice University; Willy Zwaenepoel, EPFL

  Causeway provides a general-purpose, distributed, multi-tier framework for scheduling, performance analysis, and security and access control. This project is motivated by solutions that exist for single-node systems, poor ad hoc solutions for multi-tier systems, and the lack of a general-purpose framework.

- ***PLuSH: A Tool for Remote Deployment, Management, and Debugging***

  Christopher Tuttle, Jeannie Albrecht, Alex C. Snoeren, Amin Bahdat, University of California, San Diego

  Fundamental abstractions of remote deployment include things such as abstract description language, resource discovery, resource allocation, host and environment monitoring, experiment deployment, and execution management. PLuSH is a framework of components that integrates these abstractions.

- ***Using Inferred Emergent Behavior to Automate Resource Management***

  Patrick Reynolds, Duke University; Janet Wiener, Jeff Mogul, and Marcos Aguilera, Hewlett-Packard Labs; Amin Vahdat, University of California, San Diego

  To automate resource management, we must find a system's emergent behavior from events and discover highly suspicious behavior that is different from a programmer's stated expectation, statistically anomalous, or a dominant source of delay. To find problem sources, applications are instrumented to infer a model of system behavior. Multi-resolution tracing starts with a black-box approach and then explores the benefits of additional information,

resulting in more specific, more accurate information.

- ***Using Access Logs to Detect Application-Level Failures***

  Peter Bodik, University of California Berkeley; Greg Friedman, Lukas Biewald, and H.T. Levine, Ebates.com; George Candea, Stanford University

  Sometimes it takes months or years to detect a failure in Internet services. Based on the assumption that users change behavior in response to failures, a chi-square test of access history can detect anomalous activity.

- ***A Trust-Based Model for Collaborative Intrusion Response***

  Kapil Singh, Norman C. Hutchinson, University of British Columbia

  Most intrusion detection systems emphasize detection; response is limited to blocking part of the network. This approach temporarily stops the intrusion but does not cost anything for the attacker. If network components collaborate to identify the source of attack, they can defend against it by attacking the attacker. An attacker is identified by a proof of attack using router logs of activity.

- ***The Ghost of Intrusions Past***

  Ashlesha Joshi, Peter M. Chen, University of Michigan

  There is a window of vulnerability between the discovery of a bug and the application of its patch. An administrator may not know whether an intrusion occurred in this window. An approach to this problem is to use virtual machine replay and introspection to detect the triggering of the vulnerability.

- ***SoftwarePot: A Secure Software Circulation System***

  Yoshihiro Oyama, University of Tokyo; Kazuhiko Kato, University of Tsukuba

  SoftwarePot is a user-level middleware system that provides a virtual environment "pot." The system contains a private namespace of resources and a private file tree, and it can be mapped to a real external resource.

- ***Implementing an OS Scheduler for Multi-threaded Chip Multiprocessors***

  Alexandra Federova, Harvard University and Sun Microsystems; Margo Seltzer, Harvard University; Christopher Small, Daniel Nussbaum, Sun Microsystems

  Multi-threaded chip-multiprocessors lead to contention for L2 cache. Modifying the OS scheduler to co-schedule hand-picked processes can lead to increased throughput of 27–45% and a reduction in L2 miss rate by 19–37%. Processes are characterized and profiled by predicting miss ratios by randomly sampling how often certain memory locations are reused (30% overhead).

- ***Charon: A Framework for Automated Kernel Specialization***

  Mohan Rajagopalan, Saumya K. Debray, University of Arizona; Matti A. Hiltunen, Rick D. Schlichting, AT&T Labs Research

  Charon takes a holistic systems design to combine programming languages and OS design to improve both performance and security. Charon uses binary rewriting capabilities and static analysis to achieve a reduction in memory footprint while ensuring correctness. Potential applications include synthesizing kernels for specific targets (motes, routers, cell phones), QoS, adaptation reliability, configuration checking, and bug discovery.

- ***Java in the Small: Enabling Standard Java on Embedded Devices Through Customization***

  Alexandre Courbot, Gilles Grimaud, LIFL; Jean-Jacques Vandewalle, Gemplus Research Labs

  Many embedded devices would like to run Java, but often Java does not fit because the entire JRE is too large. JITS tailors a full-fledged JRE to a specific application based on runtime usage by removing unnecessary components and reducing space overhead.

- ***Singularity: Software Systems as Dependable, Self-Describing Artifacts***

  Galen Hunt et al., Microsoft

  Singularity is a new operating system developed by Microsoft to be used for dependable systems research. Dependability, defined as behaving as expected by creators, owners, and users, is the primary goal of this project. Singularity makes configuration a first-class concept with built-in abstractions. Online and offline inspection, verification using partial specifications, and IPC via bi-directional message channels are all supported.

KERNEL NETWORKING

Summarized by Alan Shieh

- ***Deploying Safe User-Level Network Services with icTCP***

  Haryadi S. Gunawi, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison

  icTCP addresses the deployment of TCP/IP extensions. Many such extensions have been proposed in recent research. However, the transition from research to practice has been slow. Moreover, as new operating environments such as wireless networks emerge, new extensions may be needed. icTCP aims to reduce the kernel development costs of extensions by moving extensions from the kernel to user libraries, and adding a small, easy-to-implement set of kernel interfaces to enable multiple such user-level extensions. Thus, the kernel modifications are amortized over multiple extensions. Extensions written using icTCP require small amounts of kernel support, have low design and performance overhead, and are guaranteed to be TCP friendly.

  icTCP provides application read/write access to internal TCP

variables (cwnd, ssthresh). By modifying these variables, applications can modulate the send rate. These variables are virtualized in that applications are not allowed to write arbitrary values, since this would enable TCP-unfriendly flows. Instead, only those transformations allowed by RFC 2581 are allowed, and so extensions are, by definition, TCP-friendly. The icTCP virtual variables should be applicable to most TCP implementations, since the TCP variables are found in most implementations. A recent packet history may also be provided; this extension is optional, since not all implementations keep such a history, and passing this history can be expensive.

icTCP requires 316 lines of code in Linux. The effectiveness and necessity of restricting the operations on virtual variables are confirmed. Multiple TCP extensions (TCP Vegas, TCP Nice, TCP-RR, TCP-EFR) were implemented as user extensions, at smaller line-number counts than the original kernel implementations. Extensions could be combined in a stack to leverage the benefits of multiple different extensions for the same connection. Interposing a user-level extension degrades performance slightly—bandwidth is not affected at small numbers of connections, but is slightly degraded at larger numbers of connections.

George Kola (University of Wisconsin, Madison) pointed out that TCP Reno has a coarse timer resolution, while TCP Vegas has a fine-grained timer resolution; he asked how icTCP supports TCP Vegas. The response was that icTCP has more fine-grained timeout. Andrew Whitaker asked how the icTCP technique applies to non-TCP protocols, for instance, congestion-controlled UDP. The response was that the authors have explored how UDP flows can use information from TCP flows and that this has not yet been implemented. Currently, the authors have only looked at algorithmic extensions, not new protocols.

- ***ksniffer: Determining the Remote Client Perceived Response Time from Live Packet Streams***

  David P. Olshefski, Columbia University and IBM T.J. Watson Research Center; Jason Nieh, Columbia University; Erich Nahum, IBM T.J. Watson Research Center

  Response time is critical, and poor response time can have economic consequences. Also, response time must be controlled to meet service level agreements (SLAs). Improving accuracy and minimizing latency to feedback (e.g., providing online rather than offline results) could improve the efficacy of automated management systems. However, existing methodologies have shortcomings. Probing at external points is either not scalable or does not have a high sampling rate. Application-level log analysis is typically offline and does not capture all the system latencies. Instrumenting Web pages requires overhead and changing the content, and does not work for all clients.

  Ksniffer, a kernel-level latency analysis that captures the kernel and network latencies of a complete HTML page view, operates at gigabit rates on commodity hardware (e.g., relies on no driver modification and requires no special hardware) and works for all clients and all types of content, without instrumentation overhead. To minimize the persistent state of a packet, all packets are processed online, without intervening windowing or queuing. Using these techniques, ksniffer achieves low overhead while measuring response time accurately.

  ksniffer returns page view information—the latency from the initial request of the root object, to the last object on that page. ksniffer does not parse HTML to identify the last object, since parsing is too slow and the HTML does not directly correlate with the actual fetch/processing order. Instead, ksniffer uses pattern learning to determine the embedded objects for a given page using referrer fields: The referral field for a request for an embedded object (e.g., a JPG or GIF) generally points to the container. These patterns are not used for situations where container information is directly available—e.g., requests from a single HTTP/1.1 connection, referrer field available. In the remaining cases, the referrer field is inferred by matching against the pattern cache. Where appropriate, low-level TCP latencies (e.g., propagation time for the last packet, connection setup time) are added to the page view time computed from this HTTP analysis.

  The evaluation measured ksniffer under a range of experimental conditions. ksniffer results closely matched directly measured results from a modified client instrumented to directly report its perceived timeout. ksniffer correctly correlated the response time distribution within a subnet (similar distance from server), and differentiated the distribution between different subnets (different distance from server). ksniffer results also tracked the load surges in a highly variable stress test. Compared to Apache, ksniffer measured the correct response time, while Apache measured an order of magnitude lower (and incorrect) response time.

  Ilya Usvyatsky (EMC Corporation) asked, "How do content distribution networks (CDNs) affect the correlation techniques?" The response was that the only way the CDN will affect the response time is if it generates the last completing download. However, since a CDN should be much faster than the server, and runs in parallel, the last download to complete is unlikely to come from the CDN. Stefan Savage (University of California, San Diego) commented that "often there are external objects, e.g., advertising banners,

which could add significant overhead (especially when DNS is accounted for). Also layout and rendering time can dominate. These DNS/external fetches and layout issues are invisible to the server." The response was that the latency due to other Web sites can be significant, but if the critical path is not on your own server, optimizations on your server are not going to improve response time. Other tools are available for measuring end-to-end rendering. One can't measure this on the server.

- ***FFPF: Fairly Fast Packet Filters***

  Herbert Bos and Willem de Bruijn, Vrije Universiteit Amsterdam, The Netherlands; Mihai Cristea, Trung Nguyen, and Georgios Portokalidis, Universiteit Leiden, The Netherlands

  Code is available from http://ffpf.sourceforge.net.

  FFPF reexamines packet filters, since the assumptions underlying their original design no longer hold. For instance, at the time, computational speed was close to network speed; this is no longer the case. While network monitoring is important, a large fraction of traffic is unclassifiable due to shortcomings in the expressiveness of traditional packet filters. Many monitoring solutions support only slow networks, or only sample portions of the input.

  The goal of FFPF is to achieve high link rate without resorting to sampling. To allow more traffic to be classified, FFPF supports a more flexible notion of a flow as any packet stream that matches arbitrary criteria. FFPF is designed to support multiple simultaneous filters efficiently: common subexpressions of different filters are executed only once, and copying is avoided by allowing different filters to share buffers.

  To minimize bus and memory bandwidth, operations are pushed as close to the data sources as possible (e.g., executing aggregation operators on a NIC or in the kernel, rather than on the CPU or user space, respectively). For instance, a FFPF pipeline to count the number of packets in a flow would both filter and perform the count. FFPF supports multiple languages, and compiles to user space, kernel space, and network processor code (IXP1200). FFPF is faster than existing libraries; packet loss is lower than pcap, and CPU utilization is slightly lower for a single filter and considerably lower for multiple filters with common subexpressions.

FILE AND STORAGE SYSTEMS II

Summarized by Charles Weddle

- ***Energy Efficiency and Storage Flexibility in the Blue File System***

  Edmund B. Nightingale and Jason Flinn, University of Michigan

  Edmund B. Nightingale's presentation began with a discussion of ubiquitous computing—specifically, network variability, power management, and stale data. This led to the introduction of the BlueFS and the "read from any, write to many" strategy. The BlueFS's flexible cache hierarchy extends battery lifetime through energy-efficient data access, supports portable storage, and improves performance by leveraging the unique characteristics of heterogeneous storage devices.

  The presentation next discussed how BlueFS's implementation handles read from any/write to many, as well as power management, hiding device transitions, cache management, and cache consistency. The BlueFS implementation consists of a user-level daemon called Wolverine that handles reading and writing of data to multiple local, portable, and remote storage devices. It also contains a kernel module that intercepts VFS calls, interfaces with the Linux file cache, and redirects operations to Wolverine. In addition, there is a BlueFS server that stores replicated data. Most interesting is the ability of the BlueFS to hide device transitions to mask the performance impact of device power management. The BlueFS can also create device affinity so that the latest version of an object will always be cached on a particular device.

  BlueFS compared favorably to NFS and Coda in a modified Andrew benchmark, being over ten times faster than NFS and 19% faster than Coda. The evaluation showed that, because of its ability to hide access delays caused by disk power management, BlueFS can read 4k-sized files up to 60 times faster then ext2 starting from a disk in standby mode, due to the ability of BlueFS to hide access delays caused by disk power management. Someone asked whether it was assumed that a connection to the network must be present. The presenter responded that if data consistency guarantees are wanted, then a network connection must be in place.

- ***Life or Death at Block-Level***

  Muthian Sivathanu, Lakshmi N. Bairavasundaram, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison

  Muthian Sivathanu began with a discussion of how liveness information is not available in modern storage systems and how certain functionality can be enabled with this information: for example, eager writing, adaptive RAID, optimized block layout, intelligent prefetching, faster recovery, self-securing storage, and secure delete. This led into a discussion of how to make storage liveness-aware—specifically, through the two approaches taken by the authors, explicit notification and implicit detection.

  Explicit notification adds new allocate and free commands to the existing storage interface. The file

system is modified to use these commands to explicitly convey liveness information to the storage system. With implicit detection, the storage system monitors block-level reads and writes issued by the file system from underneath an unmodified interface and implicitly infers liveness information. The presentation points out that explicit notification is conceptually simple to implement but made difficult due to the asynchrony of file systems. Implicit notification can be implemented without an interface change but is fairly complex and ties the file system and storage system layers together.

The authors presented the secure delete case study they conducted to show the design, implementation, and evaluation of a secure deleting disk using both explicit notification and implicit detection. The authors chose the secure delete problem because it requires the tracking of generation liveness and provides a context in which liveness information is very important. For performance evaluation, a prototype-enhanced disk was implemented as a pseudo device driver in the Linux 2.4 kernel. Exploring the foreground performance of implicit and explicit secure delete, the authors found that the explicit implementation performs better. When asked whether the implementation duplicated file system functionality, Muthian stated that they duplicate a small amount but only on disk structures about the file system.

- ***Program-Counter-Based Pattern Classification in Buffer Caching***

  Chris Gniady, Ali R. Butt, and Y. Charlie Hu, Purdue University

  Chris Gniady began his presentation with a discussion of the buffer cache in file systems and how important the buffer cache is to performance. A key observation in process architecture is that program instructions, or the instruction's program counters, provide highly effective means of recording the context of program behavior. This led to the introduction of PC-based pattern classification (PCC). PCC identifies the access pattern among the blocks accessed by I/O operations triggered by a call instruction in the application. These pattern classifications are then used by a pattern-based buffer cache to predict the access patterns of blocks accessed in the future by the same call instruction. Chris noted that this is the first demonstration of program counter-based prediction used in operating system design.

  Chris went on to describe the PCC design and talked about the pattern classifications in PCC. There are three reference patterns that PCC uses to classify the instructions: sequential references, looping references, and other references. These classifications are used by PCC to manage future block accesses by a classified program counter. Chris then discussed the implementation of PCC, how PCC data structures capture the classifications of the program counters, and how this information is used.

  In evaluating PCC, the authors compared PCC, UBM, ARC, and LRU through trace-driven simulations. They found that PCC compares favorably to UBM, improving the hit ratio by as much as 29.3%, with an average improvement of 13.8%. PCC also outperforms ARC, with the hit ratio improving by as much as 63.4% and with an average improvement of 35.2%. Lastly, the authors found that compared to basic LRU, PCC results in an average of 41.5% reduction in the number of disk I/Os. With this disk I/O reduction, PCC reduces the average execution time of LRU by 20.5%.

;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