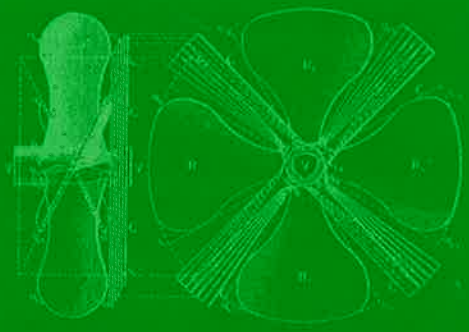
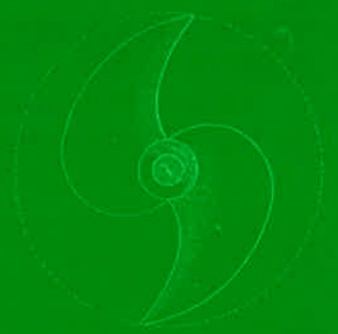
JUNE 2005 VOLUME 30 NUMBER 3

;login:

THE USENIX MAGAZINE





USENIX
The Advanced Computing Systems Association

USENIX Upcoming Events

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, 2005, Cambridge, MA, USA
http://www.usenix.org/sruti05

Linux Kernel Developers Summit 2005

JULY 18–19, 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

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

2nd Workshop on Real, Large Distributed Systems (WORLDS '05)

DECEMBER 13, 2005, San Francisco, CA, USA
http://www.usenix.org/worlds05

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

3rd Symposium on Networked Design and Implementation (NSDI '06)

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

MAY 8–10, San Jose, CA, USA
http://www.usenix.org/nsdi06

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

contents

;login:

VOL. 30, #3, JUNE 2005

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

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

;login: is the official magazine of the USENIX Association.

;login: (ISSN 1044-6397) is published bi-monthly by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

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

Periodicals postage paid at Berkeley, CA, and additional offices.

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

ROB KOLSTAD

motd

PERHAPS ALL IS NOT ROSES

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*

AS I EXAMINE THE EMPLOY- ment scene with the new SAGE salary survey, I can't help but be struck by the hottest field around: security.

I have given security tutorials and the occasional apocalyptic keynote, including warnings about Internet kidnappings. No question, the citizenry of the Net includes an unsavory element. However, I am continually frustrated at the amount of unseemly behavior that permeates our everyday computing experiences—potentially running through the entire IT industry.

Let's try to take an unbiased look at the current state of affairs.

First, let's open our emailbox. Look at that. There are about 45 legitimate messages mixed in among 403 unsolicited electronic mails. While the spam is educational in its way—what with its stock alerts, pharmaceutical announcements, beauty and body enhancement tips, reminders that I could perform a bit better in personal relationships, lottery winnings, and the occasional plea to support our brethren in Nigeria—I must confess that most of it lost interest for me after the thousandth repetition.

How did this incredibly sorry situation emerge? My guess is that no one person or entity feels sufficiently harmed by it to make it stop. Important emails buried (or hidden in a spam folder), decreased ability to find important email, and, one must believe, a certain amount of fraud: none of these is enough to warrant more than an Act of Congress that has been, in my humble opinion, not quite as effective as perhaps its framers had hoped. The cost of spam is hard to determine, since the fundamental rule of accounting ("Costs are as costs are accounted") can lead to wildly divergent answers. In my life, though, where I keep track of the number of daily invasions of my privacy and try to ensure that I don't lose any important email, spam is a truly depressing and time-consuming part of every day.

I don't think this is reasonable.

Let's fire up the Web browser now.

A pop-up! It must be really important. Why else would my time and attention be so violently taken away?

It's another home loan advertisement. I don't need another home loan. I don't need another Web camera. It's astounding the number of things I don't need. Pop-ups might be a necessary evil like advertisements on commercial television and radio, but I hate them. Turning them off in my Web browser made me feel as if my life had improved.

"My computer is so slow these days," says my friend John. A quick check reveals 600 viruses. Apparently, he tends to let Web sites entice him to download software that infects his computer. Removing all those extra features brought his performance back in line.

By the way, the use of "infection," "viruses," and the rest of the biological/health care vocabulary that surrounds this part of a security discussion strikes me as unfortunate. I think many people believe that, like human germs, computer viruses just emerge somehow in the wilderness and make their way to computers. We can't cure the common cold, and computers get sick, too. Makes sense, doesn't it?

Nothing could be further from the truth, of course. Humans engineer these viruses. The more helpful humans not only teach others how to write them but

also create Web sites that enable less skilled individuals to craft their own invaders by point-and-click. On a bad day, viruses propagate through email dramatically more quickly than through the Web.

I don't think this is reasonable.

I asked my security officer how we were doing. "Everything is great," he replied. "The firewall is stopping all the port-scans and we're all patched as of yesterday's new software."

"Port scans?"

"Sure. A few thousand times every hour, various systems try to see if any of our network services will let them break in. Sometimes we get more than one scan per second."

Now why in the world am I being scanned the as many as 100,000 times per day? Why do articles report that un-patched PCs can survive uninfected for no more than ten minutes after being connected to the Internet?

I just don't think this is reasonable. Imagine finally completing the driveway to your shiny new house. Within seconds, a host of people surround your manse rattling every doorknob, trying to open the windows, peering into your basement. Would you put up with this? Even if the people said, "We only want to look"? Of course not.

I asked the security officer about any other anomalies. "We did have a few zombied PCs, but we've reloaded them."

Zombied PCs are PCs whose resources are hijacked by someone, usually for nefarious purposes such as port scanning and spam transmission.

I don't think this is reasonable.

How did all this come about?

I suppose it came a little at a time. Cantor and Siegel opened the floodgates for "commercial use of netnews." Many well-intentioned folks didn't want to trample on free speech, so, after a few years of evolution my emailbox is inundated with offers I don't care to receive.

When challenged about innovative protocols that enabled strangers to email executable code to unsuspecting users, the world's most profitable software vendor said, "Customers are demanding these features" to enhance their experience. Those in power ignored the security folks who said, "This is a bad idea." As predicted, here we are.

Maybe the world's most profitable software company could raise its priority level for security? Oh. Never mind.

Perhaps we should heed those pundits who tell us that FireFox, Linux, *BSD, etc., really just aren't as secure as more popular software. Maybe not. My experiences simply don't bear this out.

The hell of it all is: Everyone can pay (for a spam filter, a virus scanner, a firewall, or higher costs for network bandwidth) to mitigate the security problem. Don't kid yourself if you're an end user, though; someone is paying the money on your behalf and, ultimately, it comes out of your pocket.

What are we to do?

I am afraid that these problems will jeopardize our industry's progress. To start with, not only do we need to educate ourselves and the public about the high costs of security, but we need to understand an important point: Adding security—with its expense in time and money—only gets us back to where we should have been in the first place. An ever-growing security budget yields no growth in usability (usually just the opposite) and no increase in performance or return on investment (unless you count avoiding the potential additional costs of incursion or data theft). This seems wrong.

In addition to education, I believe we need to increase social pressure to influence and to punish evil-doers who penetrate systems, steal my time, and require an entire new US$20B industry just to enable computer users to employ their systems as they were intended to be used.

Legal actions? Civil actions will not result in the deterrent effect that a round-up of a dozen spammers and system crackers might have. Put each of them in jail for a decade or two, and I imagine would-be hackers might think twice. This trend has begun with the conviction and nine-year sentence of a U.S. spammer.

Do ISPs bear any responsibility? I think so. I think they can detect some of the systems that are port-scanning and shut down their communications. Australian ISP Telstra Bigpond recently took an action like this because their resources had been strained by zombied PCs.

I am constantly amazed at the mindset of "Solve the problem close to its manifestation" rather than "Solve the problem at its source." Why aren't we going after crackers and spammers with all the force we can muster? Doesn't it matter to anyone? Is the increased cost of using computers just another small, irritating cost? Does no one realize that security problems are caused by actual people being malicious?

I don't think we're being reasonable.

letter to the editor

To Rob Kolstad:

We've long admired Rob Kolstad. He is a down-to-earth guy who is not afraid to say what's on his mind. And what's on his mind is invariably worth hearing, unless you happen to be the poor soul at the podium when he's stating his mind at the aisle microphone. However, when we read his editorial in last month's *;login:*, the one where he comes out in favor of ISPs blocking port 25 of zombie machines being used to send spam, we decided we needed to do something we thought we would never do: suggest that Rob Kolstad is wrong.

Well, maybe "wrong" is overstating it. It's not that we think the selective type of blocking Rob is advocating won't throttle the zombies the way it's supposed to do, and it's not that we think that adding this to the anti-spammers' bag of tricks is necessarily a bad thing. But neither is it true that this type of blocking or even the bag-of-tricks approach as a whole is the most effective or efficient way to attack the problem, and Rob should know that.

He should know it because, just five months ago, he sat in the audience at LISA '04 when we were presented with the Best Paper Award for our work on spam filtering. Given that we won the award, you might jump to the conclusion that we created some genius filter, or that our work was highly theoretical or very complex. Given that multiple people at the conference were heard to remark that we had "solved the problem," you might think that we'd created some magical solution, and it just hasn't reached the practical masses yet.

You might think that way, but the opposite is true. We didn't create anything. We won the award with an implementation paper. That is, we did what sysadmins are supposed to do: we read the prior work, took some commonly available components, made the adjustments necessary for our own environment, and implemented this cobbled-together solution. Our solution was to implement a simple Bayesian filter (an obsolete one, by today's standards), and—surprise!—it worked. Our paper just documents our implementation methodology and describes how you can use it at your company *right now* to effectively solve the spam problem for your user base.

Yet here is Rob Kolstad expressing his frustration at spam (a frustration we all share) and recommending, not a systematic or comprehensive solution, but yet another filter-of-the-day approach to today's most popular spammer trick, sending via zombies. Now, Rob certainly isn't alone in suggesting that we need to look to solutions other than filtering, including extreme measures such as port 25 blocks, selective and otherwise. There are a veritable circus of people like Rob—people much smarter than we are—screaming at the top of their lungs about how we can't win this fight with the "outdated" protocols we have today. They assure us that SMTP needs to be rewritten, email needs to be charged for, and access to the Internet must be censored. So how is it possible that two sysadmins at a healthcare co-op have already functionally solved this problem for their user base with a two-year-old version of a Bayesian filter? How is it possible, given that LISA '04 also had an invited Ph.D. from Microsoft claiming that "the problem with email is that it's free"?

The answer is that today's sysadmins appear to have acquired something akin to Attention Deficit Disorder wrapped in a "Somebody Else's Problem" field when it comes to the spam problem. We see the symptoms in pre-

sentations and conversations at the conferences we attend, in the papers we read, and in articles like Rob's. We hear and see sysadmins discussing federal anti-spam case law. Sysadmins demanding SMTP protocol rewrites and IETF draft acceptances. Sysadmins begging ISPs to shut off core Internet functionality for their users. Sysadmins talking about everyone in the world needing to adopt DNS hacks to send email. In short, a whole lot of sysadmins demanding that other people solve the problem for them and just tell them what to do, and a whole lot of "flavor of the week" and "reinvent everything" approaches born out of these demands. We *want* them to tell us that it isn't our problem and to explain how it's the current way things work that's broken. And if they want to charge us for some black box or for recommending that we redesign (or even turn off) core functionality, that's fine too.

What happened here? We are system administrators. It is our job to solve these kinds of problems using the tools we have available, without breaking interoperability. It's our job, and we used to enjoy it. We used to be good at it, too.

If there was a single, non-utilitarian point in our paper, a moral to our 20-page ramble, it was that we don't need censorship, FCC regulations, protocol changes, protocol kludges such as SPF, or ever-smarter learning algorithms to solve this problem. What we need is more system administrators doing competent implementations of good learning filters such as Bogofilter, crm114, and DSPAM.

We know that filters have been "beaten." There's wordlist poisoning, nefarious HTML tricks, microspam, an arms race, etc. etc. We've heard the professors and fellows, and professional smart people. We know it's impossible. The difference between them and us is that we see these attacks daily, we see them in the wild, and we've seen them in real time for two years now. And we'll summarize that experience in three words: They don't work. Despite so many people assuming these filters have been broken, the data just isn't there. To our knowledge there is not a single published paper that empirically demonstrates fatal flaws in these filters or shows them to be less than adequate at solving the problem when they are implemented using a sound methodology. Quite the contrary. Yes, they can be made to have sub-par performance when people are lazy and try to cut corners, but if the cost of solving the problem is a bit of homework and elbow grease, system administrators are the last ones who should be complaining.

Yes, we know that filters aren't the utopian solution because the spam isn't blocked at the sender. But while we argue about the ideal way to do that, users are getting ever more inundated with spam that is increasingly offensive in nature, and many of them are abandoning email altogether. We have something that may well work permanently to reduce spammers' ability to harass us and put us back on the offensive. We should take full advantage of this first and then worry about cutting spam off at the source.

Please, let's turn down the volume knob a few notches; let's help as many sysadmins as we can to get good filtering implementations; let's do our jobs. Let's see how deep the filtering rabbit hole goes for real before we chase after the next shiny idea that offers to solve the problem for us, only at the cost of a little bit of core functionality.

DAVID JOSEPHSEN AND JEREMY BLOSSER

dave@homer.cymry.org

Rob Kolstad replies:

I stand by my comment that ISPs should take measures to stop spam at its source. Filtering at the delivery point does protect end users—potentially at a high level—but it still incurs what I believe are unacceptable costs all down the line: bandwidth, CPU cycles, administrative personnel costs, and (worst of all) diluting the effectiveness of email as a powerful tool. Email's effectiveness is diluted by losing important messages (false positive detection) and wasting the time of readers (letting spam through). Of course, filtering only works for those who have filters installed. The rest continue to suffer the scourge, a scourge promulgated for no ethical reason that I can discern.

I have good results with a Bayesian filtering solution on my system, but I think that's only the beginning of a total solution.

JON FINKE

when worlds collide, 2

THE TWO-SIDED SWORD OF TECHNOLOGY INTEGRATION

Jon is a senior systems programmer at Rensselaer Polytechnic Institute, using relational databases to automate system administration and to facilitate data flow between enterprise systems. A frequent LISA author, Jon also dabbles in construction and the building trades.

finkej@rpi.edu

THE TECHNOLOGY USED TO PROVIDE telecommunication services has been evolving over the years. This often yields reduced equipment costs, increased flexibility, enhanced functions, and other good things. However, this can also drive up the complexity of our systems, increasing—or at least changing—our maintenance and support requirements, in addition to requiring expertise outside of our current staff.

This problem became apparent at our site with our new voicemail system (which uses MS Exchange as a mail store) and, to a lesser extent, our Voice over IP (VoIP) rollout.

Let me share with you a cautionary tale about the direction that early adopters are taking.

Legacy World

Looking backward for a moment, the legacy world includes equipment such as the PBX, which is based on a special-purpose computer and lots of custom hardware. This computer and its operating system had only one design function: to provide telephone service. Likewise, the voicemail system (Octel) was a dedicated, custom computer with only the single function of providing voicemail services. While these systems may have been based on general-purpose operating systems of the time, functions were much simpler then.

This approach gave the vendors a very clean environment to support and administrators a focused system to configure. They could control what operating system and application updates were needed, and, best of all, the rate of change was very low. With the exception of presumably rare bug fixes and new features, there wasn't much need for frequent updates, and those updates were driven by the application, not the operating system. In 12 years of operation, we had only a single patch for the voicemail system.

The New World

In the new paradigm (or at least the current paradigm), things have become much more complex. Our single-box voicemail system (Octel) was replaced by a collection of five machines (the Unity Voicemail server, the Exchange server, two domain controllers, and a backup server). To complicate things further, these systems are not running custom application-

specific operating systems but, rather, a general purpose operating system (Windows 2000) supplied by another vendor. We have also moved from having just a single custom application from the primary vendor to requiring additional general-purpose applications from other vendors (including Exchange, Active Directory, SQLServer, Veritas Backup, and more).

A steady stream of updates emanates from the vendors for both the operating systems and all the other applications, including the voicemail system itself. Some of these updates are bug fixes that might or might not impact our functionality; others are security patches. Unlike our old systems, these new systems operate in a network-attached world. While we may be able to use firewalls to ensure some protection, we can't always ignore the patches and bug fixes.

Our original deployment plan assumed that our new voicemail system would use the existing, supported Exchange service. Instead, we opted to install a standalone Windows domain and Exchange server. This was installed by the consultant who was assisting with the overall voicemail deployment, who left once things were up and running.

Now before you wonder how we bought such a troublesome product, I'd like to point out that we did all the proper reviews and evaluations; this all seemed reasonable going into deployment. A number of other voicemail systems we evaluated also provided unified messaging. We knew our old voicemail system was close to death, but then a 10-day outage due to a hardware failure (the 10 days was in part spent searching on eBay and elsewhere for replacement parts!) pushed us into a crash deployment project. The sudden shift from evaluation to installation led to some pushback from the department providing the Exchange email server, resulting in having to go it alone.

Challenges

The biggest challenge we face is that, as a department, we do not have any staff members with significant experience administering Windows 2000. Additionally, we face the same problem with some of the other things we need such as Exchange, Active Directory, and SQLServer. These are not simple, easy-to-pick-up systems to administer. It takes a lot of time and training for someone to become proficient in maintaining these systems. In addition to applying patches and upgrades as needed, these systems need to be monitored for problems and also tuned and adjusted to keep them operating well. This seems to be a component of the total cost of ownership that wasn't quite factored in properly in our initial thoughts.

We are currently faced with error messages about memory fragmentation on our Exchange server. Certain patches for Exchange seem to address this problem. Those, in turn, may require patches and updates to the Windows OS upon which we are running the Exchange server. There have been no updates to either system since it was originally deployed. We are also faced with the problem that attempting to get support from the vendor of the voicemail system on this problem yields as a first response, "We don't support Exchange." What is more, they won't be able to provide support for their application unless all of the other components are current with updates and patches.

In researching the memory fragmentation issue, an experienced Exchange administrator found 35 technical articles from Microsoft on this issue. However, he is not available to work on our server, and no one in our department has the background and training to readily understand these articles; they are intended for someone familiar with Exchange and Active Directory.

Conclusion

Efforts are underway to address these support issues. In the meantime, we are running a production service with unpatched, unmonitored, and unsupported software.

But what's the big picture? Is every new high-tech product going to demand its own administrator (expert) or set of experts? Will these high-feature products require the large sets of components that it appears they might? I fear at this point that we're discovering a number of hidden costs for support that our experience and background did not prepare us for. Perhaps we are paying far too high a cost for the perception of better features, but the problems remain.

HEISON CHAK

more Asterisk tricks

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

■ *heison@chak.ca*

Asterisk provides an integrated platform for many voice applications, ranging from a simple voice gateway or conference server to a full-blown PBX. The increase in Asterisk-compatible hardware being shipped by vendors will enable rapid development and deployment of voice applications to meet various needs.

IN THE FEBRUARY ISSUE OF *;LOGIN:*, a number of interesting Asterisk topics based on the dialplan and AGI (Asterisk Gateway Interface) were revealed. The dialplan (as described in extensions.conf) is the heart of Asterisk; it uses exten => and [context] to route calls, which integrates PBX (private branch exchange), IVR (interactive voice response), and external applications. This article will start off with another dialplan trick for a home office, followed by a number of effective Caller ID applications using Asterisk.

It has always been a challenge to run a PBX system in a home office environment—family members are not used to extensions and may dislike the fact that they cannot pick up just any handset when a phone rings. On the other hand, most do appreciate the privacy they get with individual extensions. Those trying to ring the house number often consider the IVR and the need to remember extension numbers as troublesome and unfriendly. To get around this, we have set up a whitelisting for people who call us frequently. If the telephone number of the caller matches an entry on the whitelist, it will ring all handsets instead of playing the IVR greeting.

```
HEISON=SIP/cisco1
CLARA=SIP/cisco2
KITCHEN=Zap/4
BEDROOM=IAX2/iaxy@iaxy
FAMILYROOM=SIP/grandstream1

HEISON_EXT=${HEISON}&${KITCHEN}&${BEDROOM}
CLARA_EXT=${CLARA}&${FAMILYROOM}&${KITCHEN}&${BEDROOM}
ALL_EXT=${HEISON}&${CLARA}&${KITCHEN}&${BEDROOM}&${FAMILYROOM}
;
; Whitelist—family members
;
exten => s,7,GotoIf($[${CALLERIDNUM} = 4161230123]?50:8) ; Dad's cell
exten => s,8,GotoIf($[${CALLERIDNUM} = 4161230124]?50:60) ; Mom's cell
;
; Calls from family rings all handsets
;
exten => s,50,Dial(${ALL_EXT},20,Tr)
;
; IVR starts here
;
exten => s,60,Background(home-greeting)
```

This scheme significantly reduces the number of complaints, while restoring a number of friendships, since it bypasses the IVR altogether for frequent callers on the whitelist. However, a caller listening to the IVR greeting can easily mistake the voice prompt for an

answering machine, because that is what most people are used to. If the caller simply hangs up, the call info is lost unless someone examines the CDR (call detail record).

Caller ID

Caller Line Identification, CLI, Caller ID, Caller Display, CID—all are different names for the service that allows one to see who is calling. Caller ID carries information about the caller, such as telephone number, name, and the network timestamp. (Note: a caller may opt to not deliver his/her identity on outgoing calls.) People often call themselves to generate CallerID in order to set the time on telephone handsets that have been moved or power-cycled.

With Asterisk allowing logic to extend beyond the dialplan, Caller ID info can easily be made available to TiVo, MythTV, any Windows PC, or Instant Messenger.

Announcement

There are off-the-shelf products that perform simple TTS (text to speech) on the incoming telephone number. They announce the caller's telephone number through built-in speakers after receiving Caller ID information. Some people find it irritating, because the device may take a few seconds to read back the 10 or 11-digit telephone number. Others find it useful, as they can be anywhere in the house and listen to the announcement before answering the call. Using the AGI and a TTS program (e.g., Festival or Cepstral), one can easily report the incoming telephone number via the system's sound card.

MythTV

The Caller ID announcement may be disturbing to people watching TV or a movie. MythTV is a suite of programs that provides a platform for watching video (e.g., TV, DVD), listening to music (e.g., CD, mp3), Web browsing, etc. Thanks to the thoughtful design of MythTV, the OSD (on-screen display) subsystem and predefined XML tags for Caller ID provide hooks in delivering information about your caller onto your television set:

```
exten => s,1,Wait(1)
exten => s,2,System(/usr/bin/mythtvosd
    —template=../programs/mythtvosd/cid.xml
    —caller_name="${CALLERIDNAME}"
    —caller_number="${CALLERIDNUM}"
    —caller_date="${DATETIME}"
exten => s,3,Answer
```

Note that the MythTV OSD can be run from a different host to your MythTV server, where you may have tuner cards installed. The multicast nature of MythTV OSD enables Caller ID to be distributed to all MythTV front ends.

As the popularity of IM (instant messaging) increases, more and more people run multi-protocol IM clients on their machine. Using AGI and Class.Jabber.php, Asterisk can send messages containing Caller ID information to an IM client supporting the Jabber protocol:

```
exten => s,1,Wait(1)
exten => s,2,AGI(jabber.php)
exten => s,3,Answer
```

jabber.php:

```
#!/usr/bin/php
```

```
<?php
ini_set('display_errors', 0 );
ini_set('include_path', '.:/usr/share/pear');
require_once('class.jabber.php');
$stdin = fopen('php://stdin', 'r');
$stdout = fopen('php://stdout', 'w');
$stdlog = fopen('my_agi.log', 'w');

while (!feof($stdin)) {
    $temp = fgets($stdin);
    $temp = str_replace("\n","",$temp);
    $s = explode(":",$temp);
    $agivar[$s[0]] = trim($s[1]);
    if (($temp == "") || ($temp == "\n")) {
        break;
        }
}

$callerid=$agivar[agi_callerid];

$JABBER = new Jabber;

$JABBER->server = "jabber.org";
$JABBER->port = 5222;
$JABBER->username = "asterisk";
$JABBER->password = "MyPassword";
$JABBER->resource = "ClassJabberPHP";

$JABBER->enable_logging = TRUE;

$JABBER->Connect() or die("Couldn't connect!");
$JABBER->SendAuth() or die("Couldn't authenticate!");

$JABBER->SendPresence();

$JABBER->SendMessage("heison@jabber.org",
    "normal",
    NULL,
    array( // body, thread... whatever
        'body' => 'Incoming call from '.$callerid ),
        NULL
        );

$JABBER->Disconnect();
exit;
?>
```

The biggest drawback to sending Caller ID info to an IM server is lag; Asterisk waits for the AGI script to complete execution before answering the call. The Asterisk dialplan was not designed to run commands in parallel. One possible work around is to use System(jabber.php&) and run the process in the background. Of course, a better solution in this case would be to run a local Jabber server on the same network to reduce the roundtrip time.

Windows PC

In case you haven't noticed, all the above examples make Caller ID available before answering the call. This brings the ability of knowing the caller's identity even before he/she gets to the IVR prompt in Asterisk. However, not everyone uses IM, and only a small percentage of people have a TiVo-like system—and some may dislike the announcement feature. To support Windows users, one might choose to deliver Caller ID information through Samba messaging or with third-party programs, such as YAC (Yet Another Caller ID program).

With the help of System() and netcat, Asterisk can push the values to a Windows machine (e.g., 10.155.1.9) listening on TCP port 10629:

```
exten => s,1,Wait(1)
exten => s,2,System(/bin/echo -n -e "${CALLERIDNAME} ${CALLERID-
NUM}'" | nc -w 1 10.155.1.9 10629)
exten => s,3,Answer
```

As soon as the TCP connection tears down (set by -w timeout_value), YAC will display the information received on the system icon tray.

Fix CIDName

Now that we know how to make Caller ID available by a number of different means, we have a fundamental problem that needs to be discussed. Having access to the caller's telephone number may be useful if you recognize the number. We often examine the Caller ID name sent by the telco, which may not be reliable.

With the PGSQL() command built into Asterisk, one can do several SQLy things, such as querying and modifying the CALLERIDNAME based on the incoming CALLERIDNUM:

```
exten => s,1,Wait,1
exten => s,2,Macro(fixcidname)
exten => s,3,Answer

[macro-fixcidname]
exten => s,1,GotoIF($[ ${LEN(${CALLERIDNUM})} = 0]?8 )
exten => s,2,PGSQL(Connect id host=... port=... password=... user=...
      dbname=...)
exten => s,3,PGSQL(Query rs ${id} SELECT fname\, lname FROM table
      WHERE number =\'${CALLERIDNUM}\')
exten => s,4,PGSQL(Fetch status ${rs} fname lname)
exten => s,5,SetCIDName(${fname} ${lname})
exten => s,6,PGSQL(Clear ${rs})
exten => s,7,PGSQL(Disconnect ${id})
exten => s,8,NoOp(${CALLERIDNAME} ${CALLERIDNUM})
```

Conclusion

The examples above demonstrate how you can use Asterisk to enhance the usefulness of the traditional Caller ID. They are all based on North American on-hook CIDs. If you look into Caller ID for call waiting or international calls, you will find that they are quite different. The Caller ID FAQ contains useful resources on the topic (http://www.ainslie.org.uk/callerid.htm).

OTHER USEFUL URLS

MythTV: http://www.mythtv.org

YAC: http://unflowerhead.com/software/yac

ANTHONY HOWE

shoot the messenger

SOME TECHNIQUES FOR SPAM CONTROL

Anthony is a Canadian software developer and sometime system administrator working in the south of France.

■ *achowe@snert.com*

EVERYONE HAS THEIR FAVORITE silver bullet for filtering unsolicited bulk email, junk mail, and spam. Unfortunately, those bullets are not perfect and can sometimes end up in your foot.

Each spam control technique has a variety of issues associated with it (as I found after having implemented nine different Sendmail mail filters [http://www.milter.info/], called "milters" in Sendmail-speak). I used them with varying degrees of success for a small ISP in the south of France. I will discuss the techniques I've tried or know about, but the following summary is by no means comprehensive.

SMTP in a Nutshell

The Simple Mail Transfer Protocol (SMTP), RFC 2821, operates based on trust (which is the cause of most of our grief) and cannot be easily replaced with something better for the foreseeable future. The IETF and their Anti-Spam Research Group (ASRG, http://asrg.sp.am/) agree on that much (as their mailing lists reveal after dedicated searching).

Briefly, an SMTP session follows these steps: connection, HELO, MAIL, RCPT, DATA, content, QUIT. Of those seven steps, only the IP address of the client connection and each valid RCPT address specified can be relied upon. Even then, the connecting IP might be questionable; because it's possibly in a dynamic IP address pool, the reverse DNS of the IP is often poorly configured or nonexistent, and now the whois information about IP and domain assignment might be restricted because of privacy concerns (RFC 3912).

As for the other steps, the HELO, MAIL, and message content can be misrepresented or faked. Even QUIT cannot be completely relied upon, since a lot of badly written mail software simply drops the connection when they are done, instead of sending the QUIT command.

Most spam filtering techniques fall into two classes: those that act on the client connection's IP address and envelope information (pre-DATA) and those that act on the message content (post-DATA). The reason I mention this is that once the DATA command is accepted by the receiving server, it is generally committed to reading the entire message until the client indicates it has finished. This, of course, consumes bandwidth and system resources, so some filtering techniques try to make a decision before accepting DATA in order to avoid/reduce more expensive forms of filtering after acceptance.

Challenge/Response

This technique looks at the sender of a message and, if he is unknown to the recipient, accepts and quarantines the message. The server then sends some sort of challenge back to the sender (who must reply, and reply correctly if it's an are-you-human test) before the server allows the quarantined message to be delivered to the recipient. A successful result is typically cached or stored indefinitely.

C/R seems to be the least welcomed of all the possible methods to filter spam. A fair amount of spam and particularly viruses fake the mail address of a real person. So one of two things happens: if the sender is known to the recipient, the message gets through without being caught; if the sender is not known, then odds are the challenge message is sent to a perfect stranger, thus creating even more spam. After a while, this gets to be really annoying for the stranger whose address has been abused. The SpamHaus DNS blacklist considers C/R systems to be just as bad as spam and will blacklist machines using C/R.

DNS Blacklists

Blacklists in one form or another have been used for filtering for a long time, but site-specific lists can be time-consuming to maintain. DNS blacklists make the process simpler by centralizing lists and exploiting DNS caching. The IP addresses of known sources of junk mail are placed on specialized DNS servers. A mail server then queries one or more of those blacklists to see whether the connecting client IP is a known spam source; if so, the connection is dropped or rejected.

Blacklists can be problematic. They first have to receive and identify junk mail or reports of such before they can list the IP address. They must be reliable, responsive, and responsible: you have to count on their information being reasonably accurate, the blacklist service should respond to valid de-list requests almost as fast as they list an IP address, and they must be consistent in their listing and de-listing policy.

Consider an ISP that, while altering the mail server configuration, makes a mistake that goes unnoticed. It is soon discovered to be an open mail relay, which is quickly listed with ORDB (http://www.ordb.org/). The ISP subsequently fixes the mistake and requests to be tested and de-listed as a "spammer." Those requests should be acted upon in a timely, preferably automated, manner. This scenario actually happened at my workplace once, but what was worse was that we consulted ORDB ourselves to reject outside sources, only to find that we had been listed and had started rejecting our own mail! ORDB is generally pretty accurate but is slow to respond to de-list requests and, as a result of a difference in time zones, we were listed as spammers for an entire business day.

Consider what happens now with spam and viruses originating from the dynamic IP pools often used with broadband. One user will have a virus-infected machine (or maybe a "zombie" computer) which sends out a stream of rubbish that results in that IP address being blacklisted. The next day a completely different user connects and is assigned that blacklisted dynamic IP; of course, the new user does not understand why he cannot send any mail. If a DNS blacklist is slow to respond or sets time-to-live values on blacklist entries too long, that IP address can remain blocked for 24 hours or more.

I've tried a variety of DNS blacklists, and the one I recommend is SpamHaus (http://www.spamhaus.org/). ORDB is good too, until it's your machine that's in the blacklist. There are many other blacklists (http://www.sdsc.edu/~jeff/spam/cbc.html), and care must be taken in choosing which to use.

Electronic Postage (Hash Cash)

Hash cash (http://hashcash.org/) is a form of electronic postage by which the sender pays postage in CPU time by performing an intensive computation that is easy for the recipient to validate. A trivial example would be that the sender computes the square root of a large number *Y* and sends the result and the number to the recipient. The recipients can validate the computation by multiplying the given answer with itself to see if it does indeed yield *Y*. Hash cash uses some properties of cryptographic hash functions to achieve the same result. The sender mints a stamp consisting of version number, timestamp, recipient, random data, and how many bits of partial-collision the stamp is claimed to have. Each message contains one hash cash header per recipient.

The idea behind this technique is that the time necessary to compute 1 or 10 hashes for a real person sending mail is insignificant, whereas the time a spammer would require to compute a hash for each recipient, when you consider that they spam thousands or millions of people, would significantly slow down their ability to send junk mail and thus increase their costs.

This is a very nice technique. The drawbacks are its status as a post-DATA verification method, and that both the sender and the receiver have to install software to mint and verify stamps. Therefore, this technique would require wide adoption before it could be used purely on its own to accept/reject messages. However, when combined with other content-filter tools

such as SpamAssassin, it can contribute a favorable score toward the acceptance of a message.

Content Filtering

There are several techniques all related to content filtering, where the entire message is received and filtered according to a set of pattern rules, Bayesian statistical analysis, and/or other techniques or external services to verify that a message is good or bad.

One content-filtering technique related to blacklists looks at all the URL domains contained within a message and looks up those domains in a DNS blacklist containing domains appearing in spam messages (http://www.surbl.org/). If a URL in the message contains a blacklisted domain, it's rejected or given a bad score, depending on the filter making the request.

Another technique has the mail server compute a checksum or signature for each message received, then queries it with services that collect signatures for all the spam that their honeypots and users report. Vipul's Razor (http://razor.sourceforge.net/) and DCC (http://www.rhyolite.com/anti-spam/dcc/) are two such services.

Bayesian analysis, as described by Paul Graham's "Plan for Spam" paper (http://www.paulgraham.com/spam.html) and his "Better Bayesian Filtering" (http://www.paulgraham.com/better.html), counts the occurrences of all the words and short sequences found in a large sample of good mail (ham) and an equally large sample of bad mail (spam). The probabilities of each of those words occurring in a spam message are computed. When a new message arrives, the words contained therein are looked up in the probability tables and the top 10 or so of the most significant are used to compute the combined probability that the message is spam or ham.

Bayesian analysis works extremely well once it's trained with very user-specific messages. For example, Mozilla and Thunderbird mail clients use Bayesian identification to delete or redirect mail to a junk folder, and they can be very accurate. But the training has to be continued as spam messages evolve, and "there be the rub." For an individual using a mail client such as Mozilla Thunderbird, it's just a simple matter of toggling an icon on incorrectly classified mail, but if you apply a global Bayesian analysis on an ISP mail server, which is possible, it can require regular maintenance, because the global statistics are not as finely tuned as they would be for an individual.

Another issue with signature and Bayesisn methods is that spammers have reacted by adding benign random words at the top or end of their messages in an effort to throw off the probabilities. Also a lot of spam has gotten shorter in an effort to provide as little content as possible from which to compute a probability and/or to throw off some pattern-based filters.

SpamAssassin (http://www.spamassassin.org/) is one of the notable mail analysis tools. Perl-based, it's a kitchen sink of spam filtering techniques, combining regular expression pattern rules, Bayesian, Razor, DCC, a variety of DNS blacklists (IP, domain, URL), hash cash, and I'm sure some other stuff I've not noticed. Using the combined techniques, it computes a score for each message, which must not exceed a site-specific threshold so as not to be classified as spam. This score can then be used by something like milter-spamc to accept, tag, redirect, reject, or discard a message.

Where I worked, we had a lot of success with SpamAssassin, but it does have its problems—most notably, it's a pig of a Perl process. The process size was about 27MB. I heard of another site that uses SpamAssassin, and their process size was about 107MB. As I recall, SpamAssassin is not threaded, because of Perl; therefore it forks when its message queue gets too long. If you have a large and active group of users (we have about 2,000), SpamAssassin can bring your mail server to a crawl when too many messages arrive in a small interval, such as with a spam or virus attack. This is one of the reasons why many sites use some form of pre-DATA filtering in combination with content filtering, to filter the easy stuff first. I've also heard of another C-based filter called Dspam that can outperform SpamAssassin, so I'm told, though I've not had a chance to look into it yet.

As mentioned above concerning Bayes training, I've learned from personal experience that SpamAssassin can also require a fair amount of hand-holding. The time might be justified if you are defending a large user base, but it appears to require just as much effort for a small number of users. SpamAssassin can retrain/autolearn itself when messages are well above or below the threshold, but when using the Bayes facility sitewide, it's a good idea to configure SpamAssassin to defer rebuilding the Bayes statistical tables on each message to a nightly cronjob, since those tables can be very large (300MB).

Date Conformance and Coherency

The milter-date filter is another very specific form of content filtering I've implemented. A mail message contains several instances of date-and-time information, such as when the message was originally written, possibly when it was resent, and when each mail server en route handled the message. Spam messages often have incorrect timestamps, appear to be too old

or too far in the future, and/or demonstrate an inconsistent timeline. The milter-date filter verifies that the date-and-time information within a message is formatted according to RFC 2822, that a message is delivered within a configurable time frame, and that the transit of a message across mail servers reflects a consistent timeline. (Note that SpamAssassin has some date-and-time verification too, though I'm not certain how specific they get.)

One problem with this method is that, surprisingly, too many people have workstations that are set with the wrong time zone, clocks off by a whole year, or similar nonsense. In the two weeks we used this milter, I saw about a dozen or so French users with their Windows workstations set to the Pacific time zone from the day they installed Windows. They just accepted the Microsoft defaults without paying attention. Also, too few servers use Network Time Protocol to keep their clock reasonably accurate, and only in Windows XP has NTP been added as part of the OS. One milter-date user reported that some of Cisco's mail servers were off at one stage and it took two weeks or so to convince a sysadmin of this fact.

Content-Transfer-Encoding Conformance and Coherency

Mail messages have a standard structure and format that is covered by RFC 2822 and enhanced by RFC 2045 and related documents concerning Multipurpose Internet Mail Extensions (MIME). A user's mail software is supposed to adhere to these documents for the formatting and transmission of mail as 7-bit, 8-bit, or binary data. A lot of spam, in particular that written in foreign languages, fails to adhere to these standards, containing unusual, often unprintable, 8-bit character codes in messages that are only supposed to contain 7-bit data for safe and correct transmission between mail servers. Many mail exchanges are very forgiving or careless in what they accept, and so this form of spam gets through. The milter-7bit is another milter I wrote (originally inspired by a mass-mailing worm) to address this class of spam. It ensures that the content of a mail message adheres to the expected or declared Content-Transfer-Encoding as described by the related RFC documents.

This technique is effective, but on its own only catches about 3–5% of spam. The most notable problem with it is that there are many mail-oriented services that fail to correctly specify or encode their mail for transport. For example, a French user might specify her real name with accented letters, but her mail client fails to use MIME word encoding in the From: header to properly specify the name. Also, some legitimate sites such as ebay.com and lhotellerie.fr send email with an explicit Content-Transfer-Encoding: header set to 7-bit, yet include 8-bit values!

Greylisting

Greylisting is a technique that uses the behavior of a normal mail server to delay the acceptance of mail temporarily. When a sending mail server initially contacts a mail exchange to deliver a message, a tuple consisting of an IP, HELO, MAIL, and/or RCPT details is recorded and the mail exchange signals the sending mail server that the message is temporarily rejected. A normal mail server will place temporarily rejected messages into a retry queue and, after an appropriate delay, attempt to resend the message to the mail exchange. The mail exchange, upon seeing the retry from the same tuple as previously recorded, accepts the message. The underlying principle here is that spammers use "mail cannons" to send as much mail as fast as they can and so will not implement a retry queue, as this is too time-consuming when sending millions of messages.

Greylisting is a nice passive technique, and proponents of the technique claim a 90% or better success rate. However, while I've not conducted any statistical analysis on it, from personal observation at my place of work I'd say those success rates are exaggerated or site-specific. And greylisting is not without its problems.

Many of our users work with the money markets and/or they treat email like FTP and instant messaging all in one. They cannot accept or understand that mail might be delayed (just try to explain RFC 2821 limits and delivery timeouts to a French user). Once the first message succeeds, though, the result should be cached for a week or two at the very least, and reset with continued correspondence, else you get an earful of grief on a regular basis.

There are also some very poorly configured mail servers, I'm guessing the pointy-clicky variety, designed and/or administered by people who have no clue about how to get a clue. The four most common issues are: (1) "Hey, I have a whizbang machine with all these CPU cycles to burn, I'll set my queue retry time to 10 seconds"; (2) "If it doesn't get through on the first try, or the second shortly thereafter, I'll wait 12 or 24 hours before retrying"; (3) "I won't retry at all" (some servers with eBay, Amazon, skynet.be, and Southwest Airlines, to name a few: see http://cvs.puremagic.com/viewcvs/greylisting/schema/whitelist_ip.txt); (4) finally, some mail systems are designed to act as a pool—I think gmail.com does this—in that any one of several machines may process the mail queue, and so messages come from different IP addresses.

Greylisting also has the problem of penalizing legitimate mailing list providers until message receivers whitelist the mailing list.

Message Limits

Message limit accounting is a facility to control the number of messages that traverse a mail exchange according to domain, sender, and/or recipient. It could be used on the outbound side, like Hotmail's daily message limits, to limit local users' consumption (particularly if they appear to be infected by a mass-mailing worm); it could be used inbound as an alternative to greylisting; or it could be enabled and disabled as needed during periods of peak mail activity, such as during a virus outbreak or spam holiday season.

I found message limits to be fine for a specific purpose; to be effective as a filtering method, however, they would require more dynamic real-time tracking of which senders are sending from where. For example, I should be able to detect if anthony@example.com attempts to send email from five different IP addresses within the space of 10 minutes, or if the same HELO argument is given for several different connecting clients, or if the same message content arrives from several different IPs.

Callback

A mail exchange that is processing an inbound SMTP transaction looks up, via the domain name system, the mail server responsible for the sender's mail. The mail exchange then opens an SMTP connection back to the sender's mail server and emulates an error return message to the sender without actually completing the transaction (i.e., never issues the DATA command). The mail server being queried normally accepts or rejects the sender's mail address in the early stages of the transaction. The idea here is that spammers use a variety of false and often invalid sender addresses in the SMTP transaction, such as false or nonexistent domains, randomly generated user names from well-known domains, facade mail systems that don't accept any mail, throw-away mailboxes that fill up with errors and replies to unsubscribe, etc.

In order for the callback to work properly, the null address must be accepted (i.e., <>) as required by RFC 2821. Many sites think they are being clever in blocking the null address to avoid spam. Often these sites fail to use more than one filtering technique and look for quick alternatives. Also, many fail to understand why the null address is a requirement in the RFCs. I have successfully educated most sites when this happens, but some others are just too enamored with their lack of prowess to admit they are wrong.

There are also those who completely disagree with this technique. They see it as some form of dictionary attack or an abuse of their mail server's resources (CPU, memory, bandwidth) to have to answer this form of automated C/R (even though the end user will never see a challenge message). One of the arguments against this form of filtering is the claim that spammers are impersonating real email addresses with ever-increasing frequency, so validating the sender's address will have diminishing returns over time and become ineffective.

Call-Ahead

The milter-ahead is a milter that implements a "call-forward" technique, which is similar to a "callback" but intended for use by mail gateways that want to verify, before the gateway accepts the message, that the recipient of a message exists on an authoritative mail store. Think of it as a poor man's LDAP. Many mail systems split the functions of mail transfer and that of storage and retrieval over two or more systems.

Historically, a mail gateway would always blindly accept and forward mail to their mail store, but spammers will often send mail to a domain using a dictionary of user names, resulting in many error message returns, which can sometimes saturate the mail gateway. Often this situation is compounded by the mail gateway queuing those useless error messages for days as they attempt to send them back to spammers who used throw-away domains or mail servers that are now off, eventually resulting in hundreds of double-bounce errors being sent to the mail gateway's postmaster mailbox.

Sequencing Delays

Sendmail 8.13 has a wonderfully simple feature, "greet pause," that catches its fair share of junk mail. When a client connection is established, the SMTP server will send back a welcome message to the client indicating its readiness. The greet-pause feature imposes a site-specified delay before it sends the welcome message. During that time, if the connecting client sends any data across the connection before it has read the delayed welcome response, Sendmail drops the connection. The concept assumes that all well-behaved mail clients must wait until after an EHLO command to determine whether the server supports pipelining. It's assumed that a lot of spam software, in an effort to be quicker and more efficient, pipeline the whole SMTP transaction from the moment the connection is established and never read a response from the server, essentially ignoring any and all errors.

While this method doesn't catch all spam, it catches a decent amount in the earlier stages of the SMTP transaction. I've often wondered why Sendmail hasn't extended the concept a little further to include the other SMTP commands. At the very least it could be applied to the EHLO command; if the mail client uses the older HELO command, then all the SMTP commands could be delayed one or two seconds before returning a response. Slowing down each step of the transaction increases the spammers' costs and reduces their efficiency.

Authorized Mail Sources

There have been several proposals put forward within the ASRG and by independents to specify a means by which a mail exchange can know whether an incoming message comes from a known and authorized source of mail. The idea here is an ISP or business declares the IP addresses of the machines that are responsible for sending outbound mail, then mail from other sources within their IP block can be considered suspect. Solutions like SPF (http://spf.pobox.com/), MTAmark, Yahoo's DomainKeys, and Microsoft's Caller ID (which merged with SPF to create Sender ID) are all variants on a theme.

SPF (classic) and subsequently Sender ID are probably the best known of these proposals. SPF uses specially formatted DNS TXT records to document sources of mail. It's a nice, simple, and elegant solution that any domain owner can manage. However, it has two significant drawbacks. First, all senders must send mail from their domain's SMTP servers, probably using SMTP authentication, which can be tricky to implement or get users to migrate to. But, more important, it breaks any form of relay or mail forwarding where the envelope sender is preserved (not sure if this applies to the Sender ID format). The SPF folks, of course, propose a solution for this: switch from mail forwarding to remailing and use something like the Sender Rewriting Scheme, VERP, VARA, etc., to rewrite the sender address. But there is a catch: these rewriting schemes can significantly increase the length of the user portion of an email address and thus break RFC 2821 maximum limits on the length of the user portion and/or overall address length. Therefore, any filtering techniques that enforce strict conformance to RFC 2821 will see a marked increase in false positives.

MTAmark is similar in nature to SPF but uses reverse DNS instead; it claims it won't break existing mail-forwarding semantics. While I haven't read this Internet draft completely, the one worry I have is that it uses reverse DNS. A domain owner does not have direct control of his IP assignment and must get his IP provider to maintain the in-addr.arpa zone for him. Some might see this as an advantage, by introducing some third-party validation. Also, some IP address assignments are resold several times over, yet the original IP provider may still control the reverse assignment. Therefore, for any legitimate business to modify their reverse DNS, they may have to go up a chain of several levels to get anything done to their in-addr.arpa entries. The other issue with in-addr.arpa is that some IP providers may not pay any attention to the merits of the request, but blindly make the changes.

I know little about DomainKeys other than to say it's patented by Yahoo and involves some form of encrypted signature added to the message headers. This means, of course, that the method is post-DATA and the mail server must accept and read the entire message before it can verify DomainKeys.

It should also be noted that authorized mail source schemes are more directed at "phishing" and "joe job" scams, where the sender of an email message is faked. By knowing the valid sources of mail, you can reject or discredit email. For example, consider a connecting client from aol.com IP space and a sender address of joe@aol.com. With something like SPF you can tell that the IP is not an official source of aol.com mail. These methods can help with spam to a degree, but that was not their original intent. The SPF Web FAQ has an interesting and lengthy section about how SPF can help with spam.

Reputation Filtering

Reputation filtering concerns a mail exchange that can query one or more third-party services for a score based on facts, trends, or reputation of a connecting mail server's IP address and/or the sender's domain. DNS blacklists are a basic form of this, but they provide only simple black/white answers. With reputation filtering, some form of history is gathered concerning the sources of mail and a score or grade is returned, providing more shades of gray.

Meng Weng Wong, of SPF fame, sees reputation and accreditation filtering as being necessary to any authorized mail source scheme, because spammers will publish SPF records, too (many already do). SPF and its derivatives are driving spammers to use their own domains, but they can still jump around the Net or discard their domains at will. But with reputation and accreditation (http://spf.pobox.com/aspen.html), you have a third party that monitors where mail is from and from whom. They can look at objective factors such as longevity, stability, and identifiability. Several of these services already exist, such as Cloud Mark, Outbound Index, and Return Path.

False Positives and Negatives

In my coverage of some of the filtering techniques in use today, I've intentionally said little about false positive (legitimate mail wrongly identified) and false negatives (the failure to identify mail as junk) for the simple reason that I have not collected any statistical data or read any detailed analysis of the techniques. Most of what I've covered here has come from personal experience, study of the techniques, and user feedback related to my milter software.

Whatever methods you end up using, be sure to read up further on the pros and cons, because there has been a lot more said about each of the methods mentioned here than I can possibly convey.

Please Don't Shoot Me

The only thing I can add to all this is, Use more than one method of filtering. Remember, a silver bullet works against werewolves, not against vampires, ghosts, demons, or spam.

THOMAS SLUYTER AND
ROLAND VAN MAARSCHALKERWEERD

when disaster strikes

CAILIN AND ROLAND DISCUSS CRISIS MANAGEMENT

In daily life, Thomas (a.k.a. Cailin) is part of a small, yet highly flexible, UNIX support department at ING Bank in the Netherlands. He took his first steps as a junior UNIX sysadmin in the year 2000. Thomas part-times as an Apple Macintosh evangelist and as board member of the J-Pop Foundation.

■ *tsluyter@xs4all.nl*

As a senior UNIX sysadmin, Roland van Maarschalkerweerd delivered input for this article, having over 20 years of experience dealing with all kinds of OSes, but over the last decade specializing in (Sun) UNIX. Besides working as a colleague of Thomas, Roland mainly enjoys bringing up four kids, providing an extra dimension in crisis-management experience.

■ *joostb8@planet.nl*

WE'VE ALL EXPERIENCED THAT SINK-ing feeling: blurry-eyed and not halfway through your first cup of coffee, you're startled by the phone. Something's gone horribly wrong and your customers demand your immediate attention!

From then on things usually only get worse. Everybody's working on the same problem. Nobody keeps track of who's doing what. The problem has more depth to it than you ever imagined, and your customers keep on calling back for updates. It doesn't matter whether the company is small or large: we've all been there sometime.

The last time we encountered such an incident at our company wasn't too long ago; it wasn't a pretty sight and actually went pretty much as described above. During the final analysis, our manager requested that we produce a small checklist to prevent us from making the same mistakes again. The small checklist finally grew into this article, which we thought might be useful for other system administrators.

Before we begin, we'd like to mention that this article was written with our current employer in mind: large support departments, multiple tiers of management, a few hundred servers, and an organization styled after ITIL (the IT Infrastructure Library). But most of the principles described here also apply to smaller departments and companies, albeit in a more streamlined form. Meetings will not be as formal, troubleshooting will be more supple, and communication lines between you and the customer will be shorter.

We have been told that ITIL is mostly a European phenomenon and that it is still relatively unknown in the US and Asia. The Web site of the British Office of Government Commerce (http://www.itil.co.uk) describes ITIL as follows:

> ITIL . . . is the most widely accepted approach to IT Service Management in the world. ITIL provides a cohesive set of best practice, drawn from the public and private sectors internationally.
>
> ITIL is . . . supported by publications, qualifications and an international user group. ITIL is intended to assist organizations to develop a framework for IT Service Management.

Some readers may find our recommendations to be strict, while others might find them completely over the top. It is, of course, up to your discretion how you deal with crises.

A Method to the Madness

The following paragraphs outline the phases one should go through when managing a crisis. The way we see things, phases 1 through 3 and phase 11 are all parts of normal day-to-day operations. All steps in between—4 through 10—are to be taken by the specially formed crisis team.

1. A fault is detected
2. First analysis
3. First crisis meeting
4. Deciding on a course of action
5. Assigning tasks
6. Troubleshooting
7. Second crisis meeting
8. Fixing the problems
9. Verification of functionality
10. Final analysis
11. Aftercare

1. A FAULT IS DETECTED

"Oh, the humanity!"

—Reporter at the crash of the Hindenburg

It really doesn't matter how this happens, but this is naturally the beginning. Either you notice something while v-grepping through a log file, a customer calls you, or some alarm bell starts going off in your monitoring software. The end result will be the same: something has gone wrong and people complain about it.

In most cases, the occurrence will simply continue through the normal incident process, since the situation is not on a grand scale. But every so often something very important breaks, and that's when this procedure kicks in.

2. FIRST ANALYSIS

"Elementary, my dear Watson."

—The famous (yet imaginary) detective Sherlock Holmes

To be sure of the scale of the situation, you'll have to make a quick inventory:

- Gather all incident cases, phone calls, and other reports related to this particular problem.
- Make a tally of the number of servers, applications, and customers affected by the problem.
- Assess the impact on each individual customer and on the company as a whole.
- Make a quick list of colleagues who are knowledgeable on the subject at hand.

Once you have collected all of this information, you will be able to provide your management with a clear picture of the current situation. It will also form the basis for the crisis meeting, which we will discuss next.

This phase underlines the absolute need for detailed and exhaustive documentation of your systems and applications. Things will go so much smoother if you have all of the required details available. If you already have things like Disaster Recovery Plans lying around, gather them now. If you don't have any centralized documentation yet, we'd recommend that you start right now to build a CMDB, lists of contacts, and so-called build documents describing each server.

3. FIRST CRISIS MEETING

"Emergency family meeting!"

—Cheaper by the Dozen

Now the time has come to determine how to tackle the problem at hand. In order to do this in an orderly fashion you will need to have a small crisis meeting.

Make sure that you have a whiteboard handy, so you can make a list of all of the detected defects. Later on this will make it easier to keep track of progress, with the added benefit that the rest of your department won't have to disturb you for updates.

Gather the following people:

- The operational supervisor or, your organization has no ops supervisor, the department head
- The resident ITIL problem manager
- The current on-call team member, meaning the one who took all the calls and who gathered the information in phase 2
- One or two people who are especially knowledgeable on the resources involved in the problem at hand (you'll select them from the short list you made)

During this meeting the on-call team member brings everybody up to speed. The supervisor is present in order to prepare for any escalation from above, while the problem manager needs to be able to inform the rest of your company through the ITIL problem process. Of course, it is clear why all of the other people are invited.

4. DECIDING ON A COURSE OF ACTION

One of the goals of the first crisis meeting is to determine a course of action. You will need to set out a clear list of things that will be checked and of actions that will need to be taken to prevent confusion along the way.

It is possible that your department already has documents such as a Disaster Recovery Plan or notes from a previous comparable crisis that describe how to treat your current situation. If you do, follow them to the letter. If you do not have these documents, you will need to continue with the rest of our procedure.

5. ASSIGNING TASKS

Once a clear list of actions and checks has been created, you will have to assign tasks to a number of people. We have determined a number of standard roles:

- One or more troubleshooters. These people perform the grunt work by going over each check or action on the list.
- One spokesperson who takes care of communications with your customers, management, and the ITIL coordinators. This person also keeps the problem record up-to-date. Basically, he's there to keep everybody out of the troubleshooters' hair, so they can do their work uninterrupted.

It is imperative that the spokesperson not be involved with any troubleshooting whatsoever. Should the need arise for the spokesperson to get involved, then somebody else should assume the role of spokesperson in his or her place. This will ensure that lines of communication don't get muddled and that the real work can continue.

6. TROUBLESHOOTING

In this phase the designated troubleshooters go over the list of possible checks determined in phase 4. The results for each check need to be recorded, of course.

It might be that they find some obvious mistakes that may have led to the situation at hand. We suggest that you refrain from fixing any of these, unless they are really minor. The point is that it would be wiser to save these errors for the second crisis meeting.

This might seem counterintuitive, but it could be that these errors aren't related to the fault or that fixing them might lead to other problems. This is why it's wiser to discuss these findings first.

7. SECOND CRISIS MEETING

Once the troubleshooters have gathered all of their data, the crisis team can enter a second meeting.

At this point it is not necessary to have either the supervisor or the problem manager present. The spokesperson and the troubleshooters (perhaps assisted by a specialist who's not on the crisis team) will decide on the new course of action.

Hopefully, you have found a number of bugs that are related to the fault. If you haven't, loop back to step 4 to decide on new things to check. If you did, now is the time to decide how to go about fixing things and in which order to tackle them.

Make a list of fixable errors and glance over possible corrections. Don't go into too much detail, since that will take up too much time. Leave the details to the person who's going to fix that particular item. Assign each item on the list to one of the troubleshooters, and decide in which order they should be fixed.

Then start thinking about plan B. Yes, it's true that you have already invested a lot of time in troubleshooting your problems, but it might be that you will not be able to fix the problems in time. So decide on a time limit, if one hasn't been determined for you, and start thinking worst-case scenario: "What if we don't make it? How are we going to make sure people can do their work anyway?"

8. FIXING THE PROBLEMS

Obviously, you'll now tackle each error, one by one. Make sure that you make note of all of the changes that are made. Once more (I'm starting to feel like the schoolteacher from *The Wall*), don't be tempted to do anything you shouldn't be doing, such as fixing other faults you've detected. And absolutely do not use the downtime as a convenient window for performing that upgrade you'd been planning on doing for a while.

9. VERIFICATION OF FUNCTIONALITY

Once you've gone over the list of errors and have fixed everything, verify that peace has been brought to the land, so to speak. Also, verify that your customers can work again and that they experience no more inconvenience. Strike every fixed item from the whiteboard, so your colleagues are in the know.

If you find that there are still some problems left, or that your fixes broke something else, add them to the board and loop back to phase 3.

10. FINAL ANALYSIS

> "Analysis not possible . . . We are lost in the universe of Olympus."
>
> —*Shirka, the board computer, from Ulysses31*

Naturally, your customers will want some explanation of all of the problems you caused them (so to speak). So gather all the people involved with the crisis team and hold one final meeting. Go over all the things you've discovered and make a neat list. Cover how

each error was created and its repercussions. You may also want to explain how you'll prevent these errors from happening again in the future.

What you do with this list depends entirely on the demands made by your organization. It could be that all your customers want is a simple email, while ITIL-reliant organizations may require a full-blown postmortem.

11. AFTERCARE

> "I don't think any problem is solved unless, at the end of the day, you've turned it into a non-issue. I would say you're not doing your job properly if it's possible to have the same crisis twice."
>
> —Salvaico, Sysadmintalk.com forum member

Even after the postmortem, you may need to take care of a few things. Maybe you've discovered that the server in question is underpowered or that the faults experienced were fixed in a newer version of the software involved. Discoveries like these warrant starting a new project. Or maybe you've found that your monitoring is lacking when it comes to the resource(s) that failed. This, of course, will lead to an internal project for your department.

All in all, aftercare covers all of the activities required to make sure that such a crisis never occurs again. If you cannot prevent such a crisis from happening again, you should document it painstakingly, so that it can be solved quickly in the future.

Final Thoughts

We sincerely hope that our article has provided you with some valuable tips and ideas. Managing crises is hard and confusing work, and it's always a good idea to take a structured approach. A clear and level head will be the biggest help you can have.

OFIR ARKIN

demystifying passive network discovery and monitoring systems

Ofir Arkin is the CTO and co-founder of Insightix (http://www.insightix.com), conducts research in the information security field, and has published research papers, advisories, and articles in the fields of information warfare, VoIP security, and network discovery & management. He is a member of the honeynet project (http://www.honeynet.org) and is a director and chairs the "security research" committee at VoIPSA (http://www.voipsa.org).

- *ofir@sys-security.com*

THE QUESTIONS OF WHAT AND WHO is on the enterprise network and what is being done on and over the network has captured the attention of many researchers interested in finding appropriate network discovery technology. Such technology would not only allow these questions to be answered accurately, completely, and in a granular fashion but would allow this information to be maintained in real time.

In the past several months a number of commercial companies have hyped a new technological solution for network discovery: passive network discovery.

This article sheds light on the weaknesses of passive network discovery and monitoring systems. While acknowledging the advantages of this technology, the article explains its shortcomings, weakness by weaknesses, and demonstrates why it is unable to deliver complete, accurate, and granular network discovery and monitoring.

Passive Network Discovery

Passive network discovery and monitoring is a technology that processes captured packets from a monitored network in order to gather information about the network, its active elements, and their properties. It is usually installed at a network chokepoint. The roots of passive network discovery and monitoring technology go back to the mid-1990s, where references regarding use of the technology can be found [1].

The kind of information collected through passive network discovery and monitoring might include the following:

- Active network elements and their properties (e.g.., underlying operating system)
- Active network services and their versions
- The distances between active network elements and the monitoring point on the network
- Active client-based software and their versions
- Network utilization information
- Vulnerabilities found for network elements residing on the monitored network

Such information can be used for the following purposes:

- Building the layer 3–based topology of a monitored network

- Auditing
- Providing network utilization information
- Performing network forensics
- Performing vulnerability discovery
- Enhancing the operation of other security and/or network management systems by providing context regarding the network they operate in (information about the network, the active elements found on the network, and their properties)

Strengths

Passive network discovery and monitoring systems have important advantages related to their mode of operation.

Real-time operation: The operation (i.e., processing received network traffic and providing relevant information) is performed in real-time.

Zero performance impact: A passive network discovery and monitoring system has zero impact on the performance of the monitored network [2]. This is because the monitored network's traffic is copied and fed into the system, the operation of which involves no active querying. This all means that passive monitoring poses no risk to the stability of a monitored network and can theoretically be installed on any network.

Data processing: Passive network discovery and monitoring systems have the ability to gather information from all TCP/IP layers of network traffic processed.

Detection of active network elements and their properties: A passive network discovery and monitoring system is able to detect network elements along with some of their properties, by observing network activity related to the network element, provided that it is receiving and responding to network traffic. This means a passive system can:

- Detect active network elements that transmit and/or receive data over the monitored network
- Detect network elements as they become active and transmit and/or receive data over the monitored network

The ability to detect active network elements based on their network activity allows passive network discovery and monitoring systems to:

- Detect network elements that have low uptime
- Detect network elements that may transmit and/or receive data only for short time periods
- Detect which network elements on the monitored network are operational and serving requests coming from network elements on other networks
- Detect active network services running on non-default ports
- Detect active client-based network software operating on network elements on the monitored network

Detection of elements behind network obstacles: A passive system can detect active network elements that operate behind network obstacles and send and/or receive network traffic over the monitored network. A network obstacle is a network element that connects multiple networking elements to a network while filtering traffic from that network to these network elements (which are logically hidden behind it). Network obstacles include a network firewall, a NAT device, and a load balancer.

Granular network utilization information: A passive solution can provide information regarding the network utilization of its monitored network link. Unlike active monitoring solutions, which only provide basic network utilization information regarding the amount of traffic observed over a certain amount of time through SNMP [3], a passive network discovery and monitoring system supplies network utilization information by observing actual network traffic. A passive system has the ability to supply more granular and detailed network utilization information (i.e., per network element, per service, etc.) than active solutions.

Network utilization abnormality detection: The ability to provide statistical information regarding network utilization information, per network element, per network service, and the ability to gather information from all TCP/IP layers, enables a passive solution to build usage profiles for any element using the network and for any service used over the monitored network. These usage profiles can later be used to detect network-related abnormalities.

Detection of NAT-enabled devices: A passive system might be able to discover network address translation (NAT)–enabled devices that operate on the monitored network and to guess the number of network devices they might hide behind them [4].

Weaknesses

Although associated with important advantages, passive network discovery and monitoring systems have a number of critical weaknesses that affect their discovery and monitoring capabilities.

What you see is only what you get: By definition, a passive system will analyze and draw conclusions about

a monitored network, its elements, and their properties from network traffic observed at a monitoring location on the network. Consequently, a passive solution cannot draw conclusions about an element and/or its properties if the related network traffic does not go through the monitoring point. Moreover, information that needs to be collected by a passive system might never be gathered, if there is no network activity to disclose the information. A passive solution cannot detect idle elements, services, and applications.

The discovery performed by a passive system will be partial and incomplete, since it is unable, technologically, to detect all network assets and their respective properties. Finally, A passive system is blind when it comes to encrypted network traffic.

No control over the pace of discovery: A passive system has no control over the type of information that passes through its monitoring point and its initiation. Statistically, certain packets might not pass through the monitoring point for extended periods of time.

Limited IP address space coverage: Lacking control over the type of information that passes through its monitoring point, a passive network discovery system can generically cover only a limited IP address space.

Not everything can be passively determined: In some cases, information cannot be discovered by using passive network discovery. Passive vulnerability discovery is a good example: not all vulnerabilities can be determined passively, e.g., the vulnerabilities abused by the Code Red worm [5], the Blaster worm [6], and the Sasser worm [7].

Incomplete and partial network topology: A passive network discovery and monitoring system gathers network topology information based on the distances discovered between network elements and the monitoring point on the network, by relying on the time-to-live field value in the IP header of observed network traffic. The time-to-live field value is decremented from its default value by each routing-enabled device that processes the IP header of the packet on its way from the sender to its destination. Some passive network discovery and monitoring systems first determine the underlying operating system of a certain network element before relying on the time-to-live field value found with network traffic initiated by this network element.

The network topology information provided by a passive system relates only to layer 3–based information, i.e., routing-based information. A passive network discovery system cannot detect the physical network topology of a network it is monitoring, for several key reasons:

- It cannot detect the network switches that operate on the network. Usually a network switch will not generate network traffic other than the spanning tree protocol, sent only to its adjunct switches.
- A passive system cannot query switches for their CAM tables, detecting which network element (or elements) are connected to which switch port.

Additionally, a passive system would supply an incomplete and inaccurate network topology map, because:

- It cannot uncover routing that does not pass through its monitoring point.
- It cannot detect other routers operating on the monitored network.
- It is unable to uncover all of the network assets operating on the monitored network.

Deployment location and the number of sensors needed: The deployment location of a passive solution determines the data quality of the network traffic it receives. Network traffic data quality is relevant to the information collection process and is maximized when the deployment location is as close as possible to the access layer (i.e., between layer 2 and layer 3). A passive system loses some of its information collection abilities when it does not observe layer 2–based traffic of its monitored network elements.

A number of passive systems must be deployed in an enterprise implementation in order to have complete coverage, with the highest quality data collection, of the enterprise networks.

Network utilization–related issue: Although it is able to receive network traffic from multiple monitoring points passively, a passive system is unable to supply per-link utilization information. Furthermore, a passive system cannot uncover communications between network elements found on the same switch on the monitoring network.

Limited service monitoring: A passive network discovery system cannot monitor service condition state transitions or uncover idle services. For example, a network service might shut down soon after serving network traffic observed by a passive system, which will remain in the dark regarding this operational state transition.

Lesser-Known and More Important Weaknesses

Some weaknesses have not had widespread publicity. Here are details about some of them, showing why they are so very important.

Cannot resist decoy and deception: Although a passive system might have some conflict resolution policies, it might be possible, although dependent on a number of parameters, to trick the system into drawing wrong conclusions about the network, its elements, and their properties, by poisoning the observed network traffic.

A passive network discovery and monitoring system's conflict resolution policies might not be effective if the monitoring location does not allow the system to receive layer 2–based traffic from the monitored network.

Influencing the accuracy of a passive network discovery and monitoring system might influence other systems, such as network intrusion detection systems (NIDS) or network intrusion prevention systems (NIPS), that rely on the data collected by the passive network discovery and monitoring system as their input.

Example 1: Changing Location Information

Discovery relies on the time-to-live field value in the IP header of observed network traffic. It is possible to trick a passive network discovery and monitoring system, under several conditions, to conclude that a certain network element is located closer to or further away from a monitoring location simply by changing the default time-to-live field value in the IP header. For example, a Microsoft Windows 2000–based networking element has the default time-to-live field value set to 128. By changing the default value to the value of 126, a passive system would identify the operating system underlying the network element as Windows, and then trust the time-to-live field value information contained within the IP header of examined packets of this network element, placing it two hops further away from the monitoring point.

Example 2: Influencing Network Traffic Utilization Information

A network element can influence network traffic utilization information by injecting bogus traffic into the network and through the monitoring location. There are many different factors that prevent a passive network discovery and monitoring system from resisting these and other more and less sophisticated types of network traffic poisoning. Among them is the inability of passive systems to validate collected information.

Denial of service & remote code execution: The need of passive systems to decode received packets passively leaves them vulnerable to DoS and remote-execution attacks, of which there have been numerous examples [8].

Conclusion

This article has examined the strengths and weaknesses of passive network discovery and monitoring technology. It has demonstrated that despite the technology's advantages, it cannot, under any circumstances, perform complete, accurate, and granular network discovery and monitoring due to limitations that directly relate to the passive nature of the technology.

REFERENCES

[1] Vern Paxson, "Automated Packet Trace Analysis of TCP Implementations," 1997.

[2] Note that it is important not to overload a network device's backplane, in case port mirroring is being used. If the network device's backplane is overloaded, the network monitored will suffer performance degradation. Another side effect would be the network device's inability to send all of the network traffic which passes through the device and needs to be monitored to the network discovery and monitoring system.

[3] For more information on active network monitoring tools, see The Multi Router Traffic Grapher (MRTG) at http://people.ee.ethz.ch/~oetiker/webtools/mrtg/.

[4] Steven M. Bellovin, "A Technique for Counting NATed Hosts," http://www.cs.columbia.edu/~smb/papers/fnat.pdf.

[5] Microsoft Security Bulletin MS01-44, Cumulative Patch for IIS, August 15, 2001, http://www.microsoft.com/technet/security/Bulletin/MS01-044.mspx.

[6] Microsoft Security Bulletin MS03-39, Buffer Overrun in RPCSS Service Could Allow Code Execution (824146), September 10, 2003, http://www.microsoft.com/technet/security/Bulletin/MS03-039.mspx.

[7] Microsoft Security Bulletin MS04-011, Security Update for Microsoft Windows (835732), April 13, 2004, http://www.microsoft.com/technet/security/Bulletin/MS04-011.mspx.

[8] For examples of DoS attacks, see "Unknown Vulnerability in the Gnutella Dissector in Ethereal 0.10.6 through 0.10.8 Allows Remote Attackers to Cause a Denial of Service (Application Crash)," CAN-2005-0009, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2005-0009; Marcin Zgorecki, "Snort TCP/IP Options Bug Lets Remote Users Deny Service," post to Snort-devel mailing list, October 2004. For an example of a remote code execution, see "Buffer Overflow in the X11 Dissector in Ethereal 0.8.10 through 0.10.8 Allows Remote Attackers to Execute Arbitrary Code via a Crafted Packet," CAN-2005-0084, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2005-0084.

ANDREW HUME

how's your OS these days?

Andrew Hume is a senior researcher at AT&T Labs. Over the past 10 years or so, he has worked on big data problems and the cluster infrastructure needed to support such applications. He believes in end-to-end checks, and in both the compelling price/performance and utter fallibility of modern PC hardware running UNIX-like operating systems.

andrew@research.att.com

IN 2003 I GAVE A KEYNOTE ADDRESS at HotOS IX about the reliability, or lack thereof, of OSes commonly used to build computing systems and clusters. I spent much time detailing examples of aberrant behavior, some of which were entertaining (if it didn't happen to you) and some just outright perplexing. While certain people, notably including those who sell software, understood my points well, I think many were not quite sure what to make of my charges.

To be truthful, I was not sure what response I wanted either. Certainly, I wanted to challenge the (often smug) complacency of the FREENIX crowd who believe in the technical superiority of their particular OS. I put forward the notion that properties that let you combine individual systems into clusters—for example, predictable and bounded behavior—used to be fairly common but nowadays seem less so.

The obvious question is, has anything changed over the last two years? The answer is clearly yes. Every release of every OS brings its own new set of "features," or bugs, as we used to call them. For example, the egregious I/O problems we had with the 2.4 Linux kernels seem to have gone away with the 2.6 kernels. Of course, new problems appear; when we are pounding away at writing to SCSI tape (at a massive 5MB/s), the buffer cache seems to vanish and becomes very slow to replenish, especially for pages read in nonsequential order. Scanning a 100MB gdbm database, which normally takes 1 or 2 seconds, starts taking anywhere from 5 to 50 minutes. I understand full well the consequences of flushing the buffer cache, but writing to a slow tape seems an inadequate reason. We find fewer bugs with each release, but the number is still decidedly nonzero.

I also want to make clear what I mean by "bug" here. I do not just mean when the OS does something wrong (more on this below), but when it does a right thing in an untimely fashion. We saw one example above, when the time needed to scan a modest database averages 1 to 2 seconds but can take 50 minutes.

So what's the problem? When we try, as my team does, to build reliable and/or highly available computing infrastructure out of nodes that are only modestly reliable, it is necessary to detect node failure. When we execute some work on a node, there is an associated time limit (called a "lease") for that work, and if the lease expires before the work completes, we assume that the node died and assign the work to

another node. This allows the general workflow to continue despite nodes failing—but we now have to parameterize the leases. If the leases are too short, there will be wasted work as we re-execute the work unnecessarily. If the leases are too long, as in the case of an actual node failure, we'll spend unnecessary time waiting for work to finish when it never will.

Another example of a time-related bug is recycling a server. Recycling a server means halting a service (which would result in the open port being closed). On most UNIX-like OSes, one can simply unmount the bind and exit; almost immediately (in a second or less), one can re-execute the server process, which will then be able to bind and proceed on. On all the Linuxen we've tried, this just fails, and we end up waiting a fairly long time before we can restart successfully (we initially wait 15 seconds, and back off exponentially to a maximum of 75 seconds). The variance of how long a wait is required (again, it seems to depend on how busy the system is) is annoying, and directly increases service unavailability.

For outright bugs, two examples come to mind. The first is the weakness of the FreeBSD SCSI system; we cannot reliably write tapes on our FreeBSD nodes (although at least we get told about the errors!). Again, the tape is slow (5MB/s) and should not be an issue, and we can reliably write them on Linux (on more or less identical hardware). Although this is annoying, it turns out reading a tape works just fine, so we're not too annoyed.

The other example is perhaps not an OS bug per se, but rather, I think, a hardware weakness, so common these days. We configure our PCs with a 3Ware controller plugged into the PCI bus, and all our disks (2–7 per node) plug into the 3Ware. Because our nodes are typically 1U systems, we need an extender board so that we can mount the 3Ware controller horizontally. It turns out that the 3Ware board is overly fussy about termination and really only operates reliably when *actively* terminated, not passively (as is the norm). We didn't really care which way it needed to be terminated; what did piss us off was the complete lack of error detection by everyone involved. No errors were logged or detected by either drivers or diagnostics. Perhaps the driver doesn't see an error, or the hardware doesn't have ECC, but this is bad.

The only diagnostic that worked was "copy 2GB to 5GB of files and checksum every copy and verify that the copies were good." In many ways, this is an admirable end-to-end test, but it also always seemed a very gross test.

Which brings me to my final thought. Many people have listened to my tales of woe, and the almost universal response is "Why are you so unlucky?" (because they not only don't see these problems, they've never even heard of them before). Certainly, I pound on my systems and work them hard. But I'm sure I'm not alone in this (although relatively few people schedule jobs in batches of 75,000–150,000, or move TBs of files around a 10 to 12-node cluster). I think the significant difference is that I *check* everything I can. All file movement is md5summed and, where plausible, we add consistency checks to verify our processing.

For example, we have a distributed logging system where the logging routine ensures that at least three systems got the log message. Each system then generates a file of the recent log messages every five minutes, and these are collected and coalesced on a central node into a single file per week. After the coalescing, we check the result by sorting all the five-minute files into an "input" pile, sorting all the weekly files we updated into an "output" pile, and then verifying that the input pile is a strict subset of the output file. You might think this a tedious, expensive check of demonstrably correct code (the shell script that does this is quite simple). But so far this check has found at least seven bugs that would have otherwise probably not been found. This includes not only bugs in the five-minute file generator, but also rude behavior by the system sort utility (returning success even though the temp file system ran out of space), and even by rcp (copying to a full file system not only returns success but also sets the file's length to the right amount, even though it failed earlier on). And thus every time I think about taking out this apparently redundant test, I think of how the system is out to get me, and I leave the test in.

So my standard answer to the question, "Why do you see so many errors?" is, "I care about the answers and check that they're right." Sometimes I wonder why more people don't have the same answer. Don't you?

ADAM TUROFF

practical Perl

DATE AND TIME FORMATTING IN PERL

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

DEALING WITH DATES AND TIMES IS A common source of needless errors. The brute-force methods of dealing with dates tend to ignore the many little details that are easy to forget. Thankfully, there are better alternatives. Using modules like POSIX or DateTime not only makes date-handling code easier to manage, but it also makes programs much more featureful and robust.

Date handling is one of those topics that is easily overlooked in many programs. The vast majority of programs I have written over the years do not need to deal with dates and times. The most common use of dates and times is simply informative, like putting a timestamp on entries in a log file:

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

use strict;

## Method 1: peppering print statements about
print STDERR "[" . localtime() . "] - process ".
    "starting\n";
## ... do stuff ...
print STDERR "[" . localtime() . "] - process ".
    "complete\n";

## Method 2: use a logging function
sub logmsg ($) {
    my $msg = shift;
    my $time = localtime();
    print STDERR "[$time] $msg\n";
}
```

Using localtime() to grab the current time is common because it's easy to use and its behavior is so simple. In order to work properly, Perl assumes a lot of context so that it can do the right thing. First, when the localtime() built-in function is called with no parameters, it assumes that you want to get the time right now and operates on the value that would be supplied by time(), a value representing the number of seconds since the beginning of the UNIX epoch.

The second piece of context here is how localtime() is used. Depending on how it is called, this function will produce either a single scalar value (a timestamp string) or a list of date-time components (seconds, minutes, hours, etc.). In the instances above, the output of localtime() is concatenated into a string, so it is used in a scalar context and would produce output like this:

```
[Fri Mar 25 12:35:28 2005] - process starting
[Fri Mar 25 12:48:02 2005] - process complete
```

Some common uses of date information are a little more involved. For example, I might want to express

the current date in YYYY-MM-DD format for archiving log files. In a shell script this is fairly trivial to do, with the date(1) utility:

```
#!/bin/sh
cd $APPHOME/logs
mv app.log app.log.'date +%Y-%m-%d'
```

In Perl, this kind of date formatting is possible, but a little more involved. To start, localtime() needs to be called in list context to convert UNIX epoch time into values such as year, month, and day:

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

use strict;

my ($sec, $min, $hour, $day, $month, $year, $wday, $yday, $dst)
   = localtime();
```

When trying to retrieve just date information, we can ignore the unnecessary values and focus on the year, month, and day values by using an array slice:

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

use strict;

my ($day, $month, $year) = (localtime())[3..5];
```

While localtime() does provide values for month and year, it mimics the format returned by the standard C library functions. Month values fall in the range 0..11, and years are the actual year minus 1900. In order to produce sensible values from localtime(), these values must be adjusted after each and every call:

```
my ($day, $month, $year) = (localtime())[3..5];
$month++;
$year+=1900;

print "$year-$month-$day"; ## format as YYYY-MM-DD
```

However, even this isn't quite correct. In order to produce a two-digit month, these values must be formatted using a function such as sprintf or printf:

```
my ($day, $month, $year) = (localtime())[3..5];

## format YYYY-MM-DD properly
printf ("%04d-%02d-%02d", $year+1900, $month+1, $day);
```

Clearly, this is a lot of work in order to do something that should be easy.

Formatting with the POSIX Module

These issues are typical of the kinds of small details that pervade handling dates and times. Thankfully, correct date and time formatting is a solved problem. C programmers may remember the strftime(3) function for handling this problem. A version of this function is available by default in Perl and is provided in the POSIX module. (This same behavior is exposed in the shell through the date(1) utility.)

Perl's POSIX::strftime() function takes a date format string as its first argument and a series of time components (seconds, minutes, . . . year, etc.) to produce a formatted date-time value. Fortunately, the order of the time values that strftime() expects is precisely the order of values that localtime() produces. Therefore, producing a date formatted as YYYY-MM-DD is as simple as:

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

use strict;
use POSIX qw(strftime);

print strftime("%Y-%m-%d", localtime()), "\n";
```

(The meaning of the formatting specifiers used in the first argument is described in the strftime(3) man page.)

Another common requirement for producing date values is to use names for months and days of the week. Frequently, programs that need to do this contain an array with the relevant names:

```
my @months;
$months[0] = "January";
$months[1] = "February";
##....
$months[11] = "December";

## Or, more succinctly:
my @months = qw(January February ... December);
```

Sadly, this is an antipattern common among programmers who do not deal with dates and times on a regular basis—that is to say, most programmers. I know I've done this more times than I care to count, and every time I feel guilty. The problem here isn't that defining an array of month or day names is necessarily wrong or bad, but it is needlessly repetitive.

Instead of redefining these lookup tables in each and every script that needs them (or, better, redefining them once in a module), why not just use the lookup tables that are already predefined in the standard C library? Here are some common formats, available through POSIX::strftime():

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

use strict;
use POSIX qw(strftime);

## Friday, March 25, 2005
print strftime("%A, %B %m, %Y", localtime()), "\n";

## Fri, Mar 25, 2005
print strftime("%a, %b %m, %Y", localtime()), "\n";
```

Creating Dates and Times POSIX-Style

Formatting times can be a tricky business, but not as tricky as performing arithmetic on dates. All UNIX date handling is ultimately done in terms of seconds since January 1, 1970, and the time() built-in function returns the current number of seconds since the start of the UNIX epoch. Figuring out the count at midnight this morning,or midnight tomorrow morning should be a simple process of adding and subtracting seconds from the current time. (The output of localtime() in list context can tell us how many hours, minutes, and seconds to fill in the missing pieces.)

For example, determining the time a few days in the past or future is just a matter of adding or subtracting multiples of the value 86,400 (that is, 24∗60∗60). While this usually works, this brute-force solution isn't quite accurate. In most time zones, there is one day a year that has 23 hours, and another that has 25 hours, marking the switch to and from Daylight Savings Time. Periodically, 86,400 seconds ago could still be "today," or it could be "two days ago." A milder version of this bug occurs when "three days after 9 a.m. Friday morning" becomes Monday morning at 8 a.m., 9 a.m., or 10 a.m., depending on the week.

There are other complications to this method. How do you obtain the time value for the beginning of next month? How do you add three months to a specific date? How do you determine "three weeks ago"?

The simple solution is to use the mktime() function, also found in the POSIX module. This function takes the same series of time components returned by localtime() and expected by strftime() and returns the corresponding epoch time. That is, the same caveats about month values being in the range 0..11 and year values being year – 1900 still apply to the inputs to mktime().

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

use strict;
use POSIX qw(mktime strftime);

## Print a timestamp for the start of 1999

print scalar(localtime(mktime(0,0,0,1,0,99))), "\n";
```

Fortunately, the values processed by mktime are not strictly limited in range. That is, mktime expects days to start at 1, seconds, minutes, hours, and months to start at 0, and so on. To ask for the time at one second before midnight midway through 2010, simply adjust the inputs accordingly:

```
##         sec min hr day mon year
print scalar(localtime(mktime(-1, 0, 0,183, 0, 110))), "\n";
```

Similarly, if I want to know what the epoch time was three weeks ago or will be three weeks hence, I can add or subtract 21 days to the current day value returned from localtime():

```
my @now = localtime();  ## get the current [sec, min, ...] values

my @past = @now;
$past[3] -= 21;  ## same time, 3 weeks ago

my @future = @now;
$future[3] += 21; ## same time, 3 weeks from now

## Print out all three dates, in chronological order
print scalar(localtime(mktime(@past))), "\n";
print scalar(localtime(mktime(@now))), "\n";
print scalar(localtime(mktime(@future))), "\n";

## Output:
Fri Mar 4 12:52:23 2005
Fri Mar 25 12:52:23 2005
Fri Apr 15 13:52:23 2005
```

(Note the switch from standard time to daylight savings time between March 25 and April 15.)

Date Handling with the DateTime Modules

For casual uses, time(), localtime(), POSIX::mktime(), and POSIX::strftime() can be used in conjunction to solve simple problems of creating and formatting time values. But there are still other problems that frequently arise when dealing with dates. One limitation of localtime() and strftime() is that they only work in the current time zone, whatever that may be. If you need to format the current time for a user in another time zone, things start to get tricky.

Thankfully, these issues are easily solved with the DateTime family of modules. As an added bonus, DateTime does away with the silliness of years being represented as "year – 1900" and months falling in the range 0..11. Here is an example of how to construct a new DateTime object that represents a single point in time:

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

use strict;
use DateTime;

## Construct an object at a fixed point in time
my $date = new DateTime (
    year => 2005,
    month => 1,  ## 1..12
    day => 1,
    hour => 12,  ## 0..23
    minute => 30,
    time_zone => "America/New_York"
);
```

```
## Construct an object for the current time
my $now = DateTime->now->set_time_zone("America/New_York");
```

The DateTime module handles a lot of details with dates and times, but it does not assume what the current time zone might be. For best results, a time zone should be specified whenever constructing a DateTime object. The time zone names that DateTime recognizes are the same ones that are found in the Olsen database, a public database of all time-zone information. (This is also the source data that is used to build the files in /usr/share/zoneinfo.) A DateTime object that is constructed without a time zone is constructed in the GMT time zone; specifying a time zone adjusts the component values accordingly.

DateTime objects can be formatted using the strftime() method, which accepts the same format strings as the POSIX::strftime() function. Because a DateTime object represents a fixed point in time, adjusting the time zone adjusts the formatted representation as expected:

```
my $now = DateTime->now->set_time_zone("America/New_York");
print $now->strftime("%c");  ## prints 'Mar 25, 2005 12:52:23 PM'

## Same time, different time zones
$now->set_time_zone("America/Los_Angeles");
print $now->strftime("%c");  ## prints 'Mar 25, 2005 9:52:23 AM'

$now->set_time_zone("Europe/London");
print $now->strftime("%c");  ## prints 'Mar 25, 2005 5:52:23 PM'
```

DateTime also handles many localization issues. For example, in French, not only are the names of the days and months different, but the standard date formats are different. Taking the same time value, we can display that time in Paris for an American viewer, a British viewer, and a French viewer. To switch the localization of a date, simply update the locale on that DateTime object:

```
## Convert to Paris time
$now->set_time_zone("Europe/Paris");

## Display, using the default (US English) localization
print $now->strftime("%c");  ## 'Mar 25, 2005 6:52:23 PM'

## Convert to a British localization
$now->set_locale("en_GB");
print $now->strftime("%c");  ## '25 Mar 2005 18:52:23'

## Convert to a French localization
$now->set_locale("fr");
print $now->strftime("%c");  ## '25 mars 05 18:52:23'
```

The vagaries of time-zone arithmetic are handled through the DateTime::TimeZone family of modules. Each of these modules define the offset from GMT and the rules for switching to and from Daylight Savings Time. The DateTime::Locale modules define the localization interfaces, and include data such as the native formats for dates and the names of the days and months. Both of these modules are installed with DateTime.

The Perl DateTime project has also built many other extensions to the core DateTime modules. Some of these modules provide calendar handling, formatting and parsing of dates, date calculations, date spans, and many other features. Sadly, the features provided by these modules are beyond the scope of this article; for more information, please visit http://datetime.perl.org.

Conclusion

Date and time handling is an area that does not get a lot of attention in many Perl programs. Using the simple and obvious brute-force techniques is actually quite complicated and very error-prone. Using a standardized library to handle dates makes the process easy and robust, whether you areusing the standard POSIX module or the DateTime modules from CPAN.

AMR EL-KADI, AHMED NASHED, KAREEM EL GEBALY, MAHMOUD ABO DAOUD, NOHA EL SHARAWY, RANIA NAZMI, AND MOSTAFA MAZEN

architecture and internal design of the AUC-Abyss Web server

Dr. Amr El-Kadi is an associate professor of computer science at the American University in Cairo. He was a member of the IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices (SEEPP). Dr. El-Kadi is a senior member of IEEE, ACM, and Eta Kappa Nu.

■ *elkadi@aucegypt.edu*

WITH THE ADVENT OF THE INTERNET, the need to deliver highly available scalable e-business systems has grown exponentially. The Web is transforming how companies transact business, communicate with their customers and business partners, and, ultimately, compete. Information technology departments are being asked to deliver and maintain systems that transact with customers around the clock, share data across the Internet, and generate large amounts of revenues. The penalty for downtime or slow response times in this environment is immense.

The term "Web service" describes specific functionality, value delivered via Internet protocols, for the purpose of providing a mechanism for another service or application to use [22]. Web services enable the specialization and reuse of traditional Web applications by exposing components of applications as Web services and enabling businesses to invoke these components. Web services will fundamentally transform Web-based applications by enabling them to participate more broadly as an integrated component of an e-business solution.

The industry is attempting to take advantage of World Wide Web Consortium (W3C: see http://www.w3c.org) and Internet Engineering Task Force (IETF: see http://www.ietf.org) standards, such as Extensible Markup Language (XML), HTTP, and Domain Name System (DNS) protocols to create specifications that define a way to publish and discover information about Web services. An example is the Universal Description, Discovery, and Integration (UDDI) specification.

Developing a scalable Web service requires developing an infrastructure to address a few fundamental challenges related to offering a service:

- Unpredictable loads, unreliable communications, and unreliable access
- Hardware scaling (i.e., the ability to arbitrarily throw hardware as scalability challenges)
- Integration (i.e., the ability to interoperate with other systems and services)

Before the Web, most communications between applications in the client-server world were synchronous. The client sent a message and then waited for the server to respond. In most synchronous situations, there was a predictable load, a simple response over a reliable communication infrastructure with reliable access (i.e., high service availability). For some situa-

tions now, a tightly coupled or synchronous service is acceptable. However, by virtue of being on the Web, the service can be exposed to unpredictable loads from unknown users over an unreliable communication infrastructure with unreliable access (i.e., can't predict the availability of other systems or Web services). On the Web, synchronous applications are often too fragile and inefficient to handle this level of uncertainty, and a loosely coupled, or messaging-based infrastructure, architecture is required.

A scenario arises where a single machine, no matter how the service is architected, does not possess the processing power required to handle Web service requests. In this scenario, Web services are deployed on a distributed multiserver architecture. As developers apply more hardware to solve critical scaling challenges, they potentially increase the complexity by introducing new factors into the architecture. Key factors such as load distribution, state management, and caching must be taken into account.

Web services are tautologically provided by Web servers, of which the Apache Web server is known to be the most widely used. The October 2003 Netcraft Web Server Survey reported that more than 64% of the Web sites on the Internet are using Apache, thus making it more widely used than all other Web servers combined. Since the 1.0 release (December 1, 1995), Apache has had a modular architecture (a feature unchanged until today [1]). Other notable Web servers include IIS (now IIS 6.0 for Windows Server 2003 [15,16]) with less than 40% of the market share, Zeus [17,18], and Flash [2,19,20]. While these Web servers outperform Apache in some aspects, they have some restrictions, such as being tied to a specific operating system or having high cost.

Three primary techniques enable the Web to handle high traffic loads: replication (mirroring), distributed caching, and improving server performance. Replication is simply duplication of Web information (either as a whole or partially) on multiple machines that either form a cluster [5] or are loosely coupled. Since any one of the machines can serve requests independently, the load of each individual server is reduced. Distributed caching includes client-side caching [6], proxy caching [7,8,9,10], or dedicated cache servers [11,12,13]. These approaches transparently cache documents closer to the clients, thereby reducing the network traffic as well as the overhead on the Web server. The effectiveness of Web caching is sometimes deemed obsolete when Web owners use cache-busting, that is, marking Web objects with a no-cache header, a technique used whenever Web owners are interested in collecting hit counts to track object popularity and usage patterns. Finally, improving server performance includes using more powerful hardware (e.g., hardware with SMP [Symmetric Multiprocessing] capability), better Web server software techniques (e.g., preforking process pools [14]), and high-bandwidth network connections [4].

Web servers, being crucial software systems, should normally benefit from advances in software engineering techniques and technology. Yet lots of software developers feel that such techniques will restrict the creativity of the developers as well as affect the performance of their products, so they elect not to use any well-defined process. We wanted to experiment with new modeling languages (such as UML), new iterative and incremental development methodologies (such as the Unified Process), new software architectures for distributed systems (such as peer-to-peer architectures), new testing techniques, and new performance evaluation methods. Open source software provides great opportunities for researchers not to start from scratch and for reuse, yet that is only possible for minor changes; making major changes to Apache was impossible, as we only have the code and neither models nor detailed designs.

Wanting to build a new Web server to experiment with all of the new technologies, we have reverse-engineered Apache and started to develop our own server. It seemed logical to us to concentrate on stand-alone servers (not clustered or

distributed servers) as a starting point, and yet ensure that their architecture would be extensible to support real businesses' ability to implement serious Web services. The beginning was a Web server called Artemis that reused many of Apache's modules and was written in C++. Artemis had good performance—close to that of the Apache version when it was developed but with a cleaner design. The goals of the AUC-Abyss project were to get rid of the many restrictions imposed by reusing Apache's modules (as they were not object-oriented by nature) and to be able to experiment freely with all of the new software development technologies.

We had as our primary objective producing a well-engineered stand-alone Web server that could outperform Apache 2.0 in comparable environments and at least support static files, fully support HTTP 1.1 requests, provide a good server-side caching mechanism, provide an efficient logging mechanism, and be both reliable and extensible.

In the following discussion we first provide some background on how Web servers function in general, while highlighting Apache's internal architecture. We then provide a summary of techniques available to handle incoming requests in parallel, detailing the model used by AUC-Abyss. The architecture of our Web server is then given and further details are revealed using static and dynamic artifacts. Before concluding, we compare the performance of AUC-Abyss against Apache.

Background

A centralized Web server, in the simplest form, could be perceived as a passive process.[1] Clients open TCP connections with the Web server and send their requested content using the HTTP protocol [1]. Since several clients may be issuing their requests in parallel, such requests are queued on the server's port. The server de-queues requests, finds the requested file, and (if found) sends an HTTP response header followed by the requested data (see Figure 1). This simple sequence is followed for satisfying static content (content that is accessible to a Web server in disk-file form). For clarity, we will not consider dynamic content (content that is generated dynamically by executing auxiliary applications) in the following discussion.

1. A passive process, as opposed to an active one, is a process that does not initiate computation. Instead, it remains dormant when there are no requests to serve and is activated by the operating system as soon as a request reaches its ports.

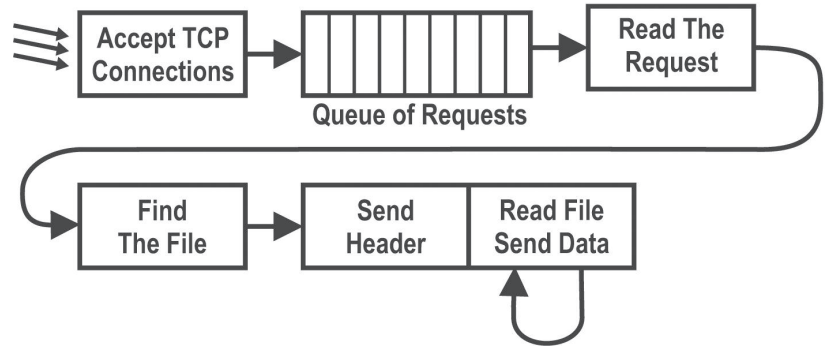


FIGURE 1: HANDLING OF HTTP REQUESTS

The open source model has stimulated the development of Apache functions by many volunteer programmers (and even recently by IBM), resulting in a fairly rapid pace of functional enhancements. Apache's modularity permits its users to pick and choose modules to fit their requirements. It is claimed that it can serve a large number of concurrent clients, limited only by the underlying hardware and operating system. The hybrid threading/multiprocess model increases its scalability. The Apache Portable Runtime layer (APR) means it can run at its

best on multiple platforms, which now include everything from common UNIX variants, the Microsoft Windows family, and NetWare, to OS/2.

The server can be configured easily (statically or dynamically) either by editing text files or by using one of the many available GUIs. Its modularity allows many features that are necessary within special application domains to be implemented as add-on modules and plugged into the server. To support that, a well-documented API is available for module developers. Its modularity and the existence of many free add-on modules make it easy to build a powerful Web server without having to extend the server code. Using many of the available server-based scripting languages, Web-based applications can be developed easily. When using scripting languages or add-on modules, Apache can even work with other server applications such as databases or application servers. Therefore, Apache can be used in common multi-tier scenarios. Additionally, Apache is completely HTTP 1.1 compliant in both of the current versions, and it also supports the HTTP compression enhancement tool, thus saving bandwidth, a feature heavily used by Google in running Apache as its Web server.

Since our target was to perform better than the Apache Web server, it was logical to attempt to understand the issues that affect its performance [4]. The single biggest hardware issue affecting Web server performance is RAM. A Web server should never have to swap, since swapping increases the latency of each request beyond a point that users consider "fast enough." This causes users to hit "stop" and "reload," further increasing the load. Too many clients attempting to connect to an Apache Server at one time can spawn child threads to the point where the need for memory swapping leads to a severe performance problem.

Concerning the issue of process creation (thread spawning), Apache's available threads are not always sufficient to accept all incoming requests, so constant per-second spawning is required. In addition, Apache's parent and children communicate with each other through something called the scoreboard. Ideally, this should be implemented via shared memory, which is the case for those operating systems that the designers had insight into. The remaining implementation of the Apache Web server defaults to using an on-disk file, which is both slow and unreliable (and less featured).

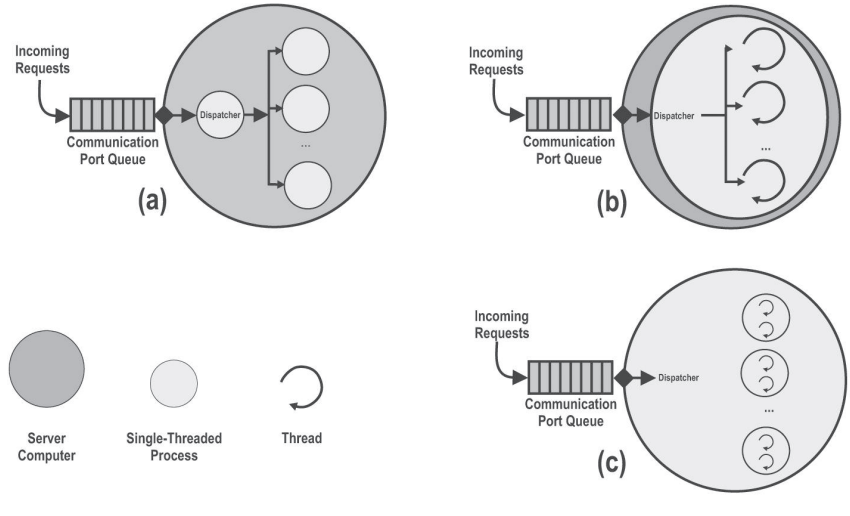


FIGURE 2: THREE MAIN PROCESSING MODELS

Parallel Handling of Requests

Before we detail the AUC-Abyss Web server architecture, it is important to discuss how clients' requests are actually handled. Web servers parallelize the han-

dling of clients' requests to exploit interleaving processing with I/O requests, thus reducing overall response time.

Two main issues greatly affect the performance of a Web server: the processing model, and the pool size behavior [3]. The processing model describes the parallelism adopted by the Web server in terms of processes and/or threads, while the pool size behavior specifies how the number of processes (or threads) varies over time in response to workload.

Three main options for a processing model are used by Web servers: the process-based model, the thread-based model, and a hybrid model (see Figure 2). In the process-based model (see Figure 2a), a dispatcher process receives requests from the queue and sends them to single-threaded processes for handling. In the thread-based model (see Figure 2b), a single multi-threaded process receives requests from the queue and assigns each to one of its own threads (lightweight processes). The hybrid mode has a dispatcher (which is a single-threaded process) that receives requests from the queue and sends them to multi-threaded processes (see Figure 2c). Each of these models has advantages and drawbacks, summarized in Table 1.

	Pros	Cons
Process-Based	Stability of the system. If a process goes down, the only effect would be the failure of the client being served by that process, without any other effect on the system.	High cost for creation and destruction of processes. Memory requirements are much less because threads share the same address space. Huge context-switching overhead.
Thread-Based	Memory requirements are much less because threads share the same address space. Spawning threads within the same process is much more efficient than spawning new processes. Much efficient inter-thread communication through the use of the shared address space.	Not as stable as the process-based model; one malfunctioning thread will take down the whole server.
Hybrid	It combines the pros of both models; if a thread crashes, it would take down the process that created it and its sibling threads. This means that some of the clients will be disconnected but not all of them.	

TABLE 1: PROS AND CONS OF EACH OF THE THREE MAIN PROCESSING MODELS

Any of the three processing models has one of two options for coping with varying workloads by controlling how the number of processes (or threads) varies over time (i.e., pool size behavior). In the first approach, a static pool is used in which a fixed number of processes (and/or threads) are created at startup. As a request arrives, it is more likely that this request will find a process already spawned ready to serve it, so no time is wasted on spawning or killing processes or threads. However, when the load on the Web server is low, many processes (or threads) will remain idle (wasting a lot of cycles and forming more switching overhead, especially if they use polling and do not block). Also, if there are p processes (or threads) already created and a request arrives finding other p requests being processed, the request will wait in a queue. As the workload for the Web server increases, the queue will get longer, increasing the response

time. In the second approach, a dynamic pool is used in which the creation and destruction of processes (and/or threads) varies dynamically according to the workload. This means that when the load is increased, more requests will be processed concurrently and the queue will be reduced. Yet at the same time, the dynamic creation and destruction of processes (and/or threads) does introduce an overhead for the server machine.

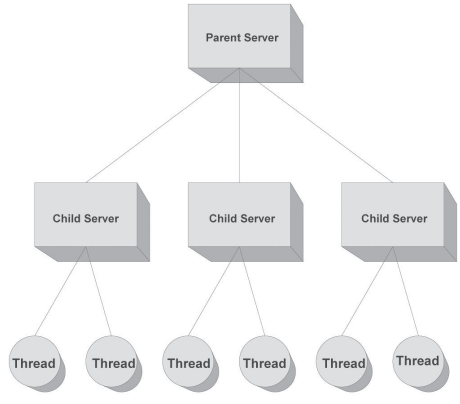


FIGURE 3: AUC-ABYSS PROCESS MODEL

We have decided to use the hybrid processing model for AUC-Abyss. In this model, processes as well as threads do not need to interact at all, since each thread serves a different request independently of the others. As our Web server starts, a single-threaded process root loads a configuration file to set up and configure its consequent operations. It starts to allocate and initialize pre-configured memory in RAM for its use. Once this phase is concluded, the root process forks another process and kills itself. That new process is the parent server (see Figure 3), which, in turn, is responsible for forking more child servers, depending on the workload and the condition of the currently running child servers. (See Figure 4a for the use-case diagram to serve an HTTP request.) Each child server spawns a number of threads by which the requests are actually handled (see Figure 4b for the use-case diagram of child servers). The fact that each thread handles a request independently improves the robustness of the server by reducing the likelihood of events that may cause a systemwide failure. However, the acceptance of new requests is synchronized through a mutual execution mechanism in order to make sure that each request is served only once. Furthermore, the child servers communicate with the parent server through the scoreboard, a file in which each child saves its current state.

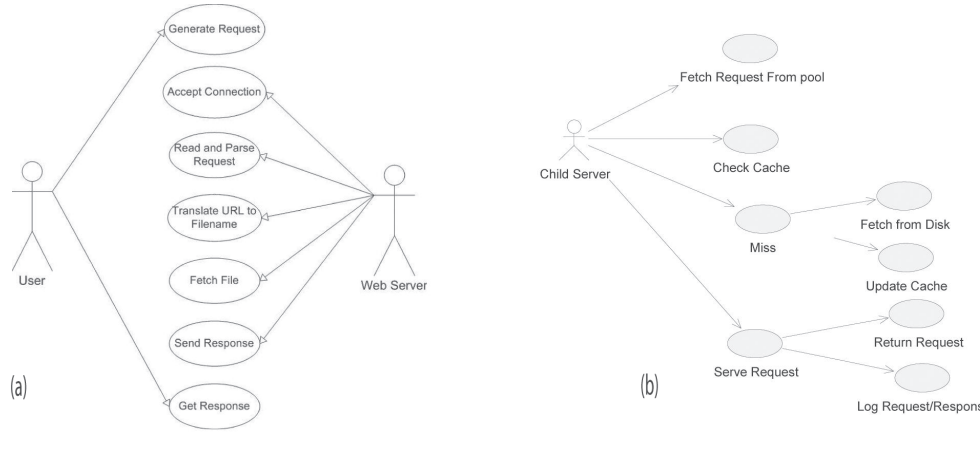


FIGURE 4: MAIN USE-CASE DIAGRAMS

Server Architecture

AUC-Abyss is based on a hybrid architecture in which many threads, spawned by child server processes, serve requests in parallel; this might dramatically decrease Web server performance, since file I/O operations are the most expensive operations a thread can perform as it competes with other threads (at least to access the log file). Thus, our primary concern was to find a way to reduce the cost of repeated I/O operations that occur after each request-response cycle. Our initial solution was to create a single global memory base buffer in which we could temporarily store the logging information and to call a periodic dumping function which would copy all this information to the log file on the hard disk. This was thought to improve performance, since it reduced the overhead of opening and closing a file stream for each single served request by buffering a large number of entries together and writing them in one chunk.

Our second concern was to maintain the performance of the Web server: the level at which a Web server is able to perform under a certain workload should remain constant, even while the server is flushing logging information. This raised a problem: once the flushing operation is underway, the memory buffer locks and cannot be accessed, the threads therefore are all forcibly put to sleep (since they cannot log after serving), and for a few seconds the server grinds to a complete halt. This was unacceptable. We came up with a twofold solution to this problem. First, and most important, it was decided that the flushing operation could not be performed by the threads themselves, since this reduces performance dramatically. Either the parent server process would perform this operation or a new twin thread (also known as a shadow thread) would be spawned to perform this operation, then die. Second, we decided to implement a mirror buffer, which performs exactly like the base buffer but is used as a backup. When the base buffer is being flushed, logging is automatically shifted to the mirror buffer and vice versa; thus request handling will never stop. Concerning the dumping of the logs from the memory, once a buffer is filled, the logging mechanism is responsible for spawning a twin thread for transferring this information onto the disk concurrently with the normal operation of the Web server, and therefore the performance of the Web server is not affected (except for using up some extra clock cycles). This twin thread is considered a twin to the threads in the process that made the final entry into the logging buffer. The parent of this twin thread is random and is not specified a priori.

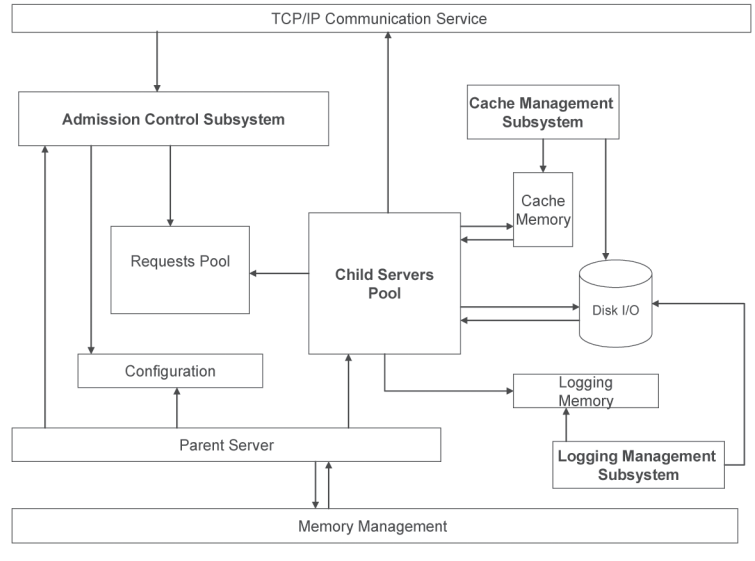


FIGURE 5: AUC-ABYSS ARCHITECTURE

Figure 5 shows the architecture of the AUC-Abyss Web server. Memory management is the most important component; it interacts with or is used by all the different entities in order to utilize memory effectively, avoiding leakage and reducing system calls. The configuration layer is responsible for the different types of configuration we handle in the system and is saved in a text file that is read when the server starts to boot. It deals with almost all the other entities that exist in our Web server.

The parent server controls the whole Web server. Since AUC-Abyss is a preforking Web server, the parent server is responsible for creating the child servers (processes) which spawn multiple threads that become responsible for handling the request-response cycle. The parent server communicates with the memory management, the configuration, and the child servers.

Once the child servers are forked, they handle all the request-response cycles of the system. The request enters through the TCP/IP interface and the admission control and is kept in a queue in the request pool. Child servers take requests and handle them; they look for them in the caching subsystem and, when done, log the operation through the logging subsystem. Finally, the TCP/IP layer provides the basic networking capabilities that the Web server needs to process its different activities. It interacts with the parent server and the request-response layer.

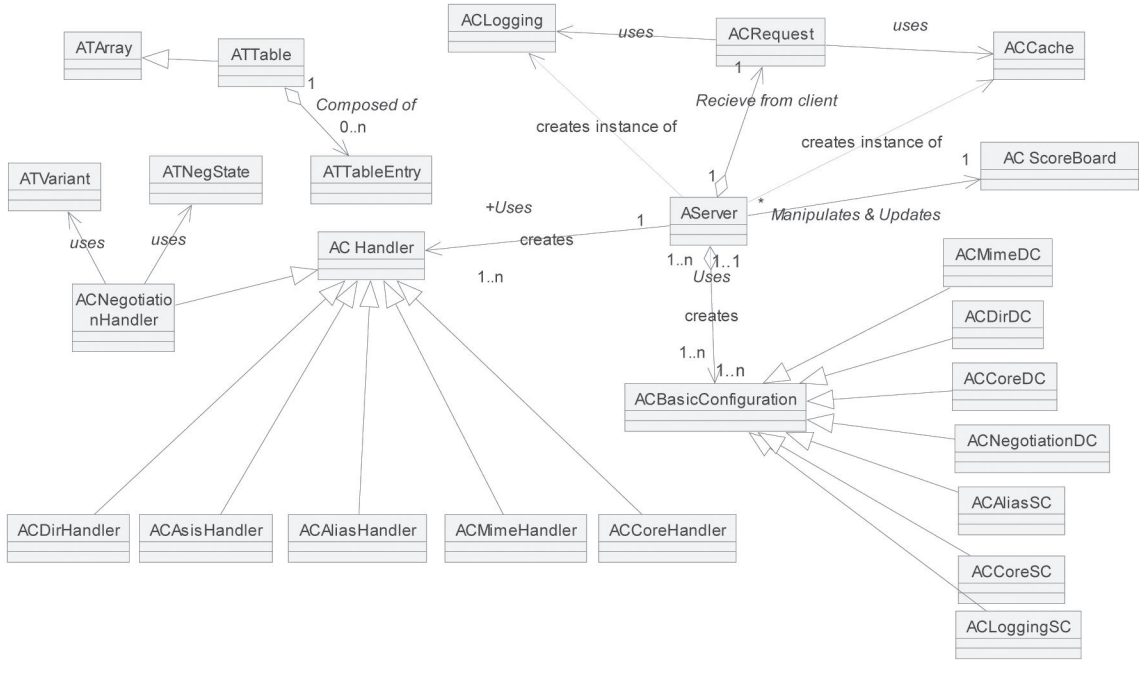


FIGURE 6: GENERAL CLASS DIAGRAM

Logical View

Our general class diagram more or less maps the system architecture, using several components (see Figure 6). The configuration subsystem consists of an abstract virtual class (ACBasicConfiguration); all other classes implement this basic class. This fosters extensibility by future addition of subconfiguration components as long as such components inherit from the abstract class and implement all its virtual functions. Two main ideas drive the existence of an abstract class. First of all, reusability suggests that all common operations and attributes be grouped into one class. Secondly, such a class facilitates adding new directives if the administrator desires. For example, an admin can create a new class in which each directive is saved with a function pointer to execute

when this directive is met in the configuration file. This class can be dynamically linked to the server and provide the extra functionality needed.

The ACCache class communicates with ACRequest during the service of the request; the logging class deals with the ACRequest class as well. The rest of the classes (AServer, ACRequest, and ACHandler) handle the request-response cycle. ACScoreboard is responsible for keeping track of the spawned processes and threads (child servers).

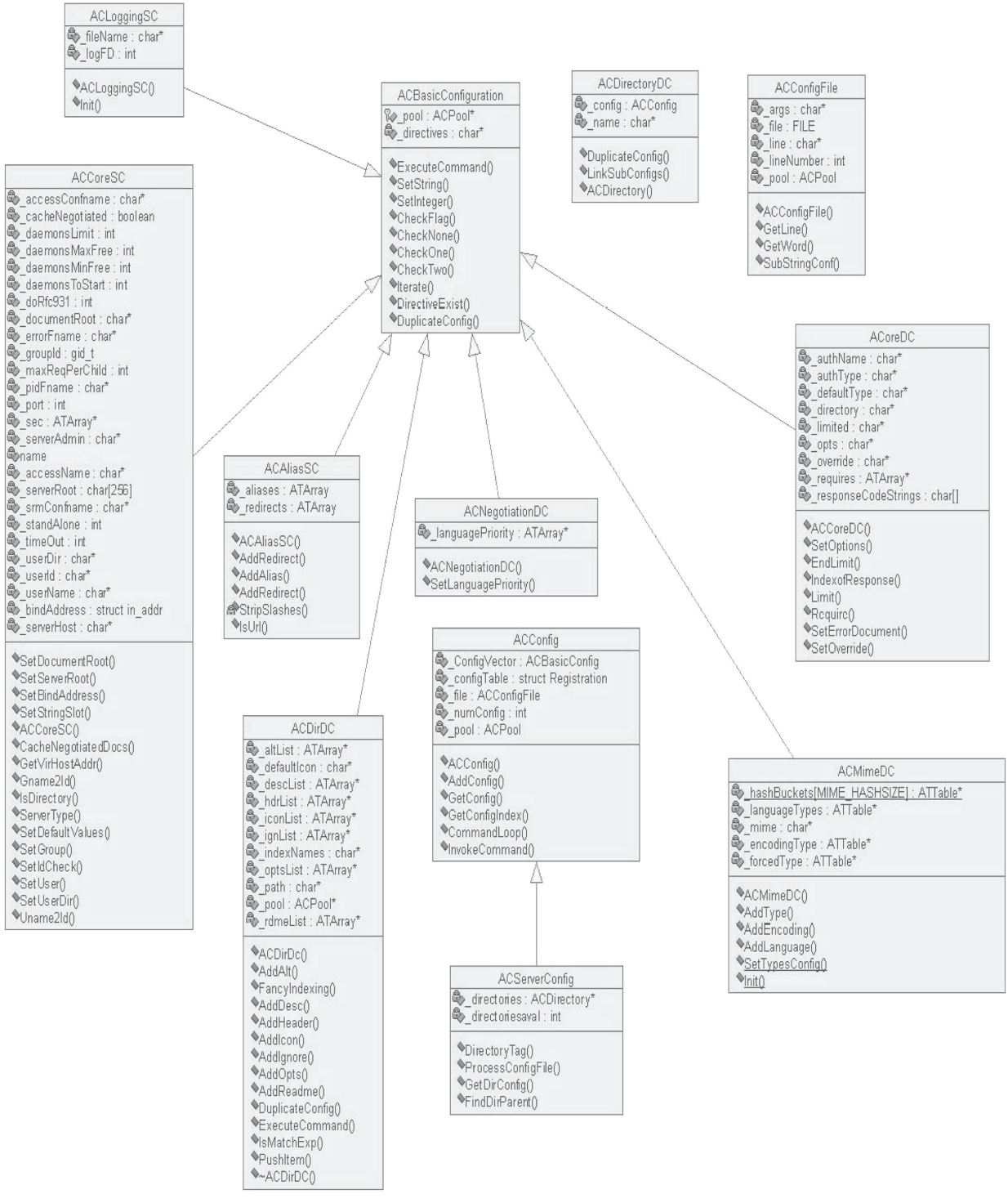


FIGURE 7: CONFIGURATION SUBSYSTEM CLASS DIAGRAM

The server cannot function properly without a configuration. There are two types of server configuration: per server, or per directory. The configuration is saved in a text file which is read when the server starts up. The configuration layer is responsible for the configuration of the server (see Figure 7). This subsystem deals with almost all the other entities that exist in the Web server. The naming schema is uniform to differentiate between various subconfigurations: those belonging to per-server configuration end with SC, whereas those belonging to per-directory configuration end with DC. The server configuration can either be a preloaded or an extra per-server configuration that can be added later on. The per-directory configuration can be either default or special.

For per-server configuration, the parent server, after creating the pool of memory, interacts with the configuration object to initialize the server configuration needed to process the various servers' activities. For example, the port number and document root are set by this object, among other variables needed by the server to start processing different requests. A preloaded per-server configuration is the basic configuration; the extra per-server configuration gives the user the ability to customize some of the configurations after the default configuration has been loaded.

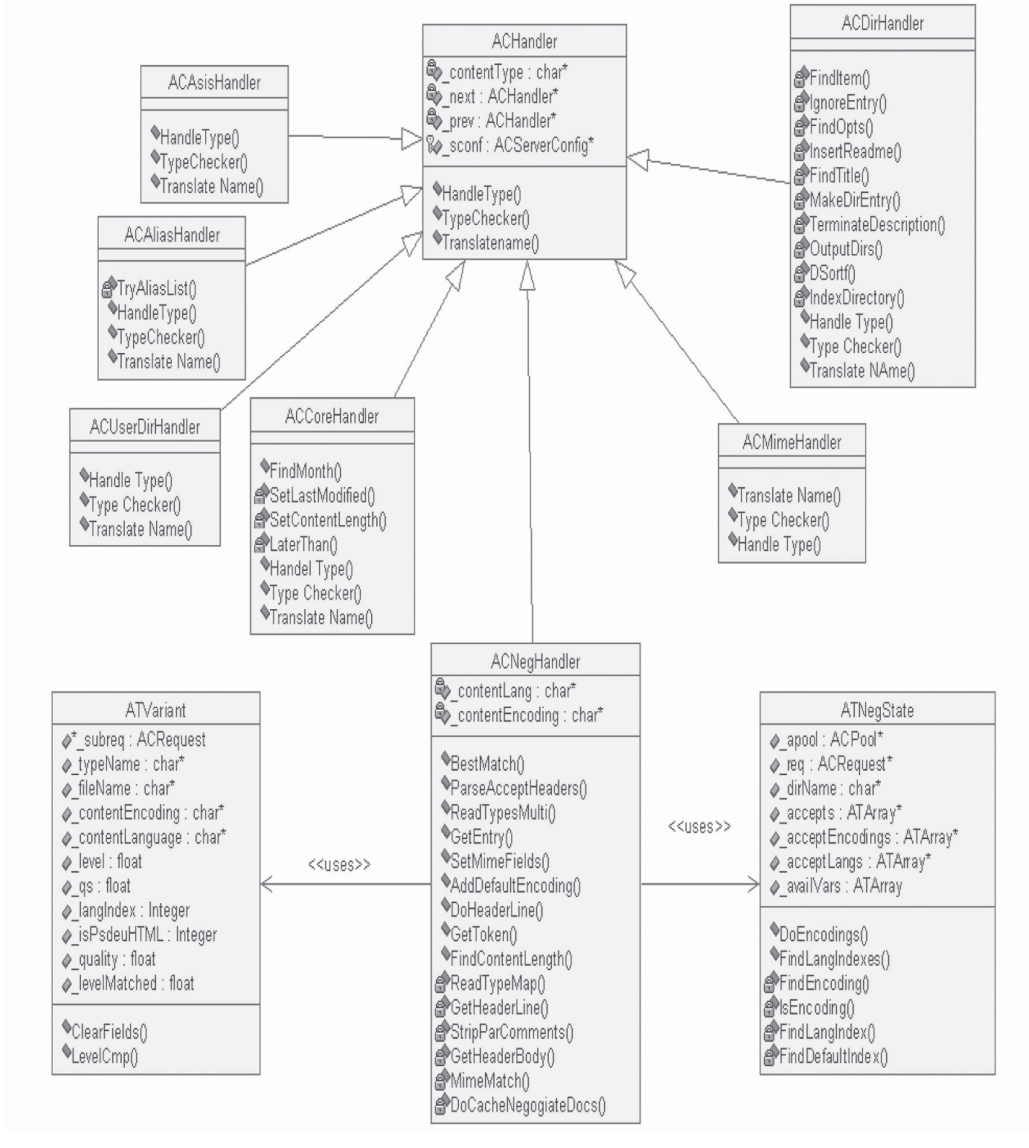


FIGURE 8: HANDLERS CLASS DIAGRAM

Per-directory configuration is responsible for initializing different directories existing in the system, both default and special configurations. Once again, the parent server interacts with the directory objects (directory, MIME, core, and negotiation) to initialize the basic configuration of each. To set up the default per-directory configuration, the parent server initializes first the server configuration and then the necessary directory configuration. This configuration is to be used when no special values are specified. For example, /usr/local/etc can use the default configuration /usr/local directly. the user can save special directory configurations, after the default initialization. These values will of course override the default values.

AUC-Abyss uses different handlers for different types of requests. The ACHandler base class contains the basic methods needed by all handlers to respond to requests. Each type of handler is a class inheriting from the base class. There are two other classes that are not handlers per se in the diagram: ATVariant encapsulates information about a particular variant, and ATNegState deals with the state of negotiation. AUC-Abyss deals with handlers as a linked list containing an instance of each. Once the server receives a request, it will loop over all these handlers and check the content type to know which one will be used in handling this specific request. This design is useful for reusability purposes.

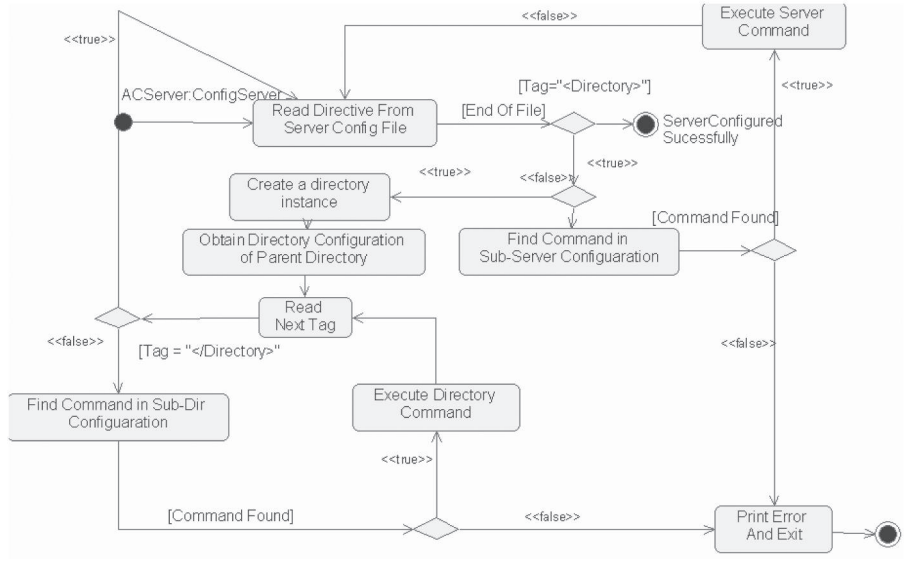


FIGURE 9: CONFIGURATION STATE DIAGRAM

Dynamic Behavior

As the root server process is responsible for the configuration of the server, it starts by executing `configServer()`. This function opens the configuration file and reads the directives (see Figure 9). Reaching end of file means the configuration was completed successfully. Each directive is read and then searched for in the command list. If there is a match, the function associated with the directive is executed and the process is repeated for the next directive. If it is not found, an error message is printed and the system is exited.

One important directive is "directory," which specifies that the user wants to create a special directory that should have a specific per-directory configuration. The server creates a directory instance and then initializes it with the default configuration by checking its parents' configuration until we have a complete directory configuration. The server then proceeds by reading another directive from the file, but this time the search is made in the table of subdirectory configurations. If a match is found, the associated function is executed and the

process is repeated until the closing tag for the directory is reached. This indicates that the special directory configuration has ended, and the server goes back to configuring the server.

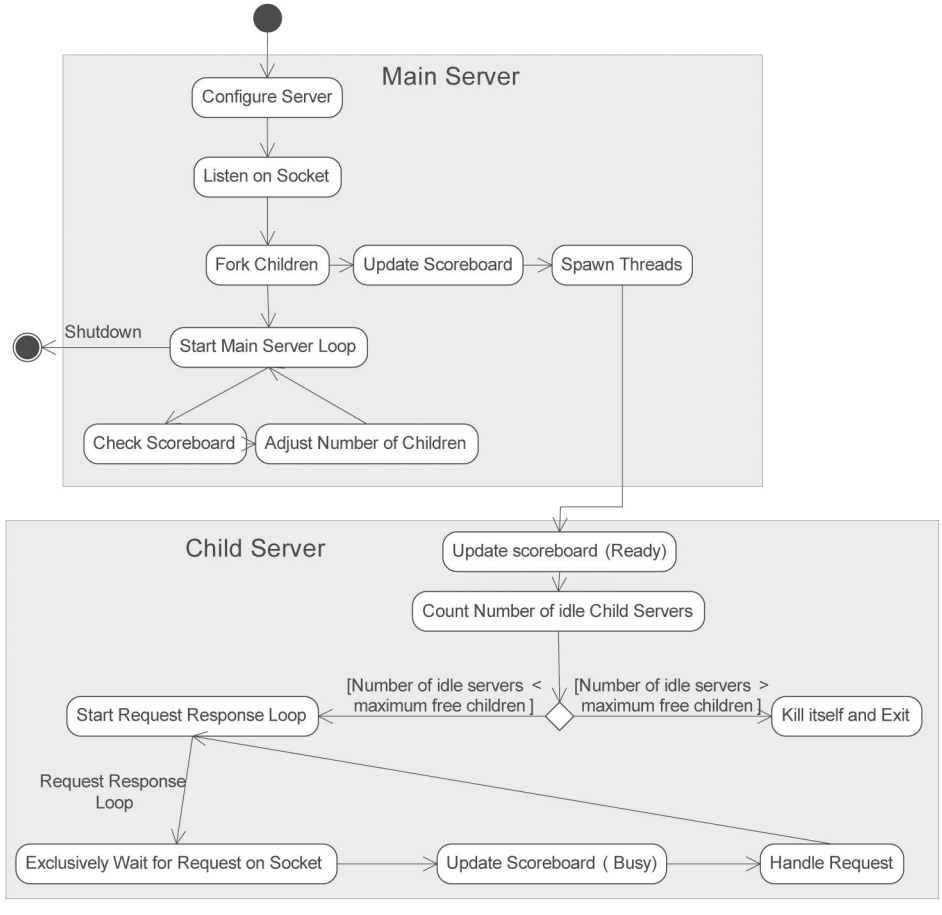


FIGURE 10: MAIN SERVER AND CHILD SERVER STATE DIAGRAM

Figure 10 shows the overall state diagram of the Web server. It starts the execution of the root process that is responsible for the entire configuration. It then opens a socket and keeps on listening for the incoming requests. Then it forks a number of processes, each of which in turn spawns a number of threads and updates the memory with the new status. Each of these threads represents a child server class responsible for handling a request-response cycle. When a thread is spawned, it updates its status in the scoreboard to "Ready" and counts the number of idle children. If that number is more than the maximum number allowed, the thread is killed and exits the connection; otherwise it starts the request-response loop. At the beginning of this loop, the thread (or child) is solely responsible for waiting on incoming requests on the socket; when a request arrives it is accepted and then handled. It then goes back to the start of the request-response loop. Meanwhile, the main server is in another infinite loop, maintaining the child statuses and the scoreboard. Note the hierarchy of the system: the main server represents the parent responsible for a number of child servers, whereas each child server is responsible only for handling a request.

Request-Response Sequence

The request-response cycle (shown in Figure 11) begins with an instance of the AServer class (see Figure 12). This object is the main core controller of the Web server and begins by creating instances of ACCache andACLogging, followed by the ACHandler classes. The AServer is solely responsible for handling the request-response cycle. It creates an ACRequest object and then loops infinitely, waiting for a connection from a client through the WaitForConnection() function. Once it is notified of a pending request connection, it proceeds to open the input/output connections through the ACRequest object initialized earlier through the sockets (OpenInputConnection() and OpenOutputConnection() functions). It is then concerned with processing the request by reading the request, parsing the URL, and obtaining the content type of the request through three functions: SendBasicHeader(), SendHttpHeader(), and SendFile(). Once this is done, the ACChildServer asks the ACServer controller to get the handler suitable for handling this type of request. Thus, the handler type is returned through a response message. The child server then proceeds to ask the ACHandler object to handle this specific type of request.

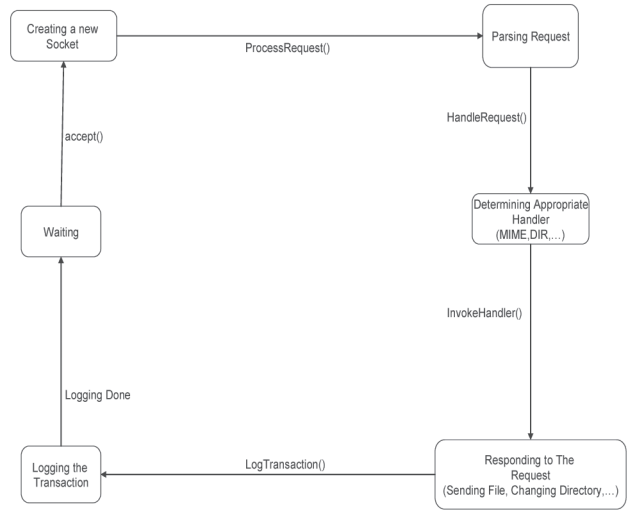


FIGURE 11: TYPICAL REQUEST-RESPONSE CYCLE

In responding, the ACHandler will first look up this request in the cache memory by using the lookup() function in the ACCache instance that was initialized earlier by the ACServer. A response is then sent back to the handler from the ACCache containing a pointer to the requested data in memory if it is found there. If not, a pointer to its location on the hard disk is returned. The child server uses this pointer to perform the response part of the cycle. This is achieved by sending the Basic Header, the HTTP Header, and the file itself through the functions previously mentioned in the ACRequest object back to the client through the OpenOutputConnection() of the socket. After the response is delivered, the child server is responsible for storing the transaction in the log buffer. This is accomplished by using the insert() function in the ACLogging object.

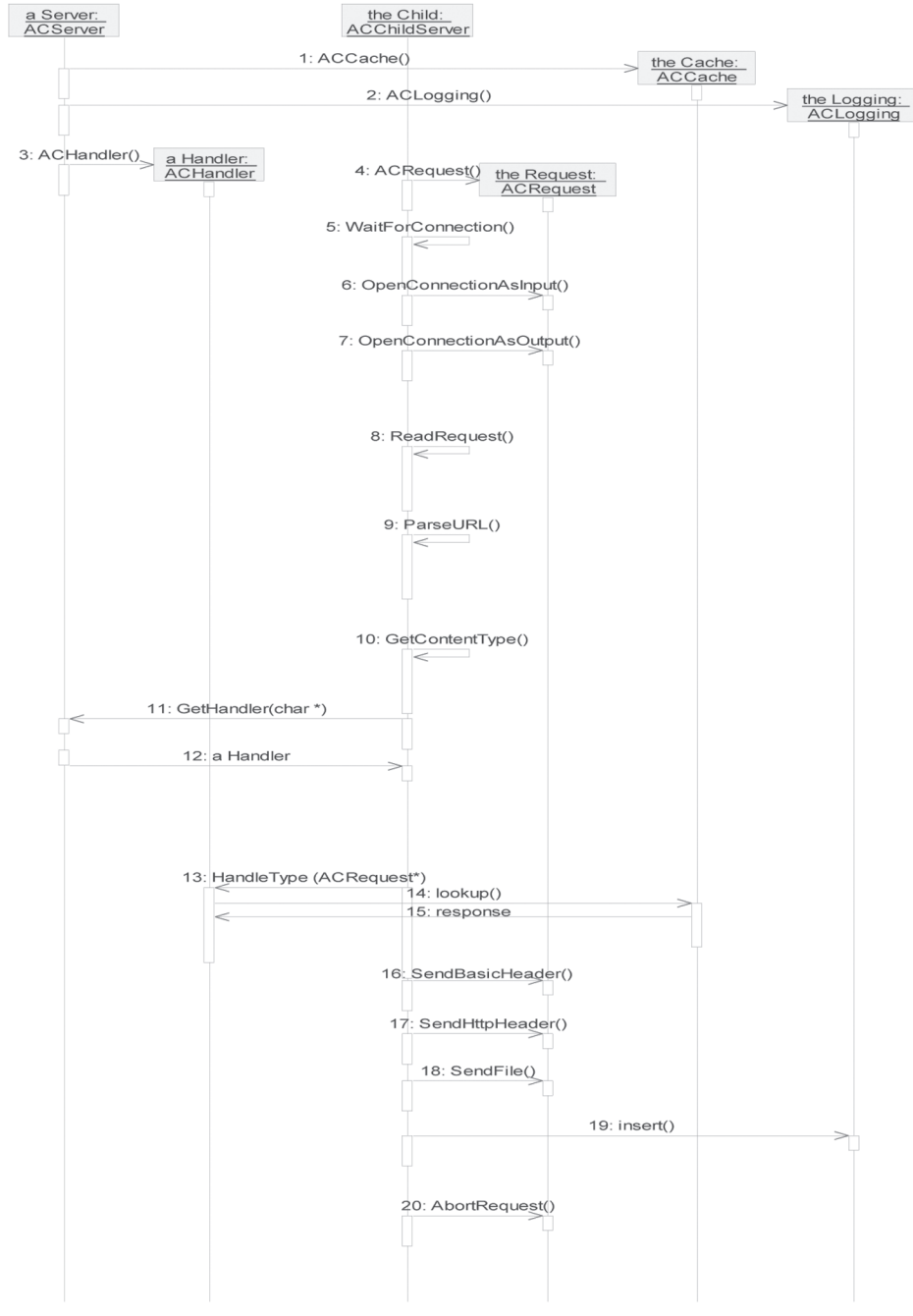


FIGURE 12: REQUEST-RESPONSE SEQUENCE DIAGRAM

ALICE 15 Kbytes	Test 1	Test 2	Test 3	Average	% Difference
AUC-Abyss	2	2	2	2	21.70%
Apache	2	3.3	2	2.4333333	-21.70%
The number of connections was 50,000 with a rate of 100 connections per second. The Net I/O of the network was 12.5 Mbps.					
BECKY 256 Kbytes	Test 1	Test 2	Test 3	Average	% Difference
AUC-Abyss	26	28.2	30.6	28.266667	2.90%
Apache	27.9	31.5	28.1	29.166667	-2.90%
The number of connections was 20,000 with a rate of 44 connections per second.					
The Net I/O of the network was 92.3 Mbps.					
CANDY 512 Kbytes	Test 1	Test 2	Test 3	Average	% Difference
AUC-Abyss	50.8	49	48.8	49.533333	1.55%
Apache	52.8	49.2	48.9	50.3	-1.55%
The number of connections was 10,000 with a rate of 22 connections per second. The Net I/O of the network was 92.3 Mbps.					
DOROTHY 1 Megabyte	Test 1	Test 2	Test 3	Average	% Difference
AUC-Abyss	95.8	97	97.2	96.666667	0.30%
Apache	97.2	97	96.7	96.966667	-0.30%
The number of connections was 5000 with a rate of 10 connections per second. The Net I/O of the network was 88.3 Mbps.					
EDITH 2 Megabyte	Test 1	Test 2	Test 3	Average	% Difference
AUC-Abyss	189.7	190.7	189.6	190	0.14%
Apache	189.6	190.6	190.6	190.26667	-0.14%
The number of connections was 2500 with a rate of 5 connections per second. The Net I/O of the network was 88.3 Mbps.					

TABLE 2: REQUEST-RESPONSE TIME TEST RESULTS

Performance Evaluation

The first set of performance evaluation benchmarks was concerned with testing the request-response time of both Apache 2.0 and AUC-Abyss. We used httperf (a standard benchmarking tool for evaluating Web servers), taking three samples for every file size and repeating the experiment for five different file sizes (15KB, 256KB, 512KB, 1MB, and 2MB), each test running for 10 minutes. By looking at the results (see Table 2), we notice that in small file sizes we outperformed Apache by a fairly obvious margin, but as file sizes grew in size, the performance of Abyss converges with that of Apache. This is due to network saturation as the network I/O reaches its maximum. It's also important to note that a file of 15k is the most common file size requested on the Internet in general, and in that case AUC-Abyss outperformed Apache by an average of 22%.

We then moved on to the width tests, focusing on evaluating the server's concurrency performance when numerous small files were requested. Concurrency is of vital importance here, especially for Web sites visited by millions of people (e.g., google.com or hotmail.com). These tests were carried out on both AUC-Abyss and Apache. Once again, each test was repeated three times and performed using four small file sizes (20B, 1KB, 2KB, and 4KB). All tests were at a constant rate of 2500 connections per second, which worked out to 150,000 total connections.

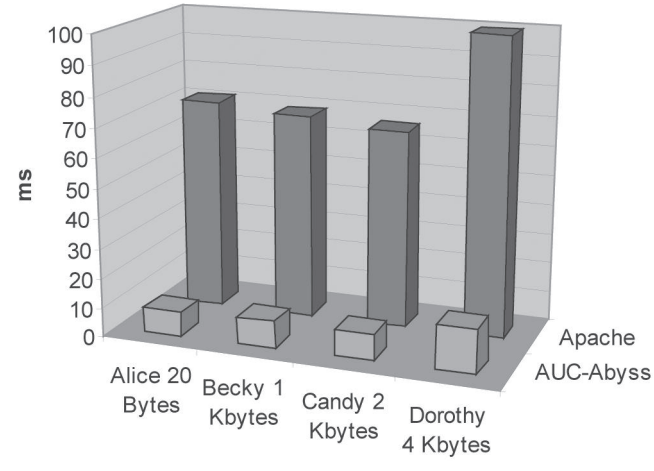


FIGURE 13: AVERAGE RESPONSE TIME

The results indicate that Abyss handled concurrency dramatically better than Apache (see Figure 13). It is important to note that Apache was inconsistent in its maximum number of concurrent users, which is unacceptable in an enterprise situation.

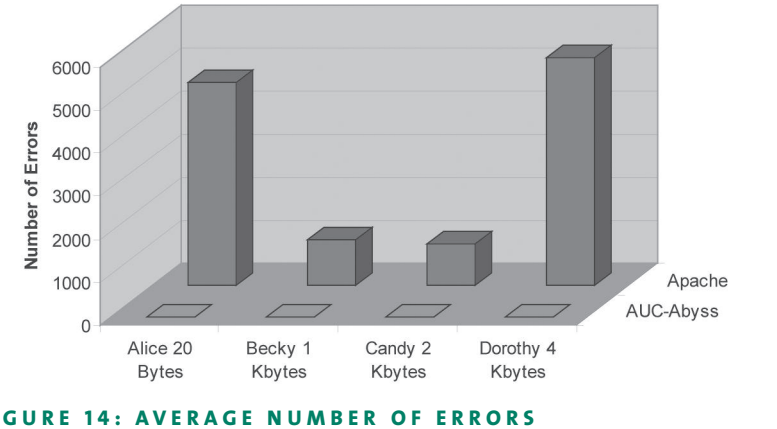


FIGURE 14: AVERAGE NUMBER OF ERRORS

The error performance of AUC-Abyss was outstanding, as it produced the least number of errors (zero errors, in fact) in comparison with Apache (see Figure 14). As a result of the number of errors, we can conclude that the Apache experienced denial of service whereas AUC-Abyss was still up and running.

Conclusion

We can safely state that our project met its goals. We have designed and implemented an extensible Web server using state-of-the-art software engineering technology that is on a par with the most widely used Web servers. But this is just the starting point. Much more work needs to be done to make AUC-Abyss as powerful as other Web servers. As yet, it does not provide sophisticated

admission control, it only supports static files, and it is a stand-alone, non-portable server.

Most Web server architectures reject excess requests without discriminating between different resource bottlenecks, or they use only one indicator for overload, often CPU utilization. Hence, they cannot take the potential resource consumption of requests into account, but have to reduce the acceptance rate of all requests when one resource is over-utilized [21]. Both high CPU utilization and dropped packets on the networking interface can lead to long delays and low throughput. Other resources that could be controlled are disk I/O bandwidth and memory. The admission control mechanism adaptively determines the client request acceptance rate to meet the Web servers' performance requirements, while the load balancing or client request distribution mechanism determines the fraction of requests to be assigned to each Web server (in the case of a distributed-based Web server architecture).

Adding admission control over and above basic load balancing reduces workload, increases server performance (faster response to users' requests), and maximizes the usefulness of server arrays. It is observed that admission control ensures that throughput is maintained at the highest possible level by controlling traffic to the Web servers when the servers' resources are approaching exhaustion. By controlling traffic before resources are exhausted, the chances of server breakdown are minimized, and hence system sanity and graceful degradation, in the worst case, are guaranteed. In addition, if admission control allows a user access to a Web server, the user will receive continuing priority access to server resources, thereby ensuring that the service a user perceives is maintained at an acceptable level.

To conclude, admission control plays a crucial role in ensuring that the servers meet users' quality-of-service requirements while maximizing site availability and preventing server congestion/failure during heavy traffic. Our next step for AUC-Abyss is to add admission control. The fundamental question here is, Is admission control really necessary in Web server systems? In response, we note that there are two ways to increase overall user utility, namely, increasing server (farm or cluster) capacity or implementing intelligent traffic management mechanisms. Our experiments show that we can utilize resource-based admission control to avoid over-utilization of critical Web server resources. We may also provide service differentiation using token buckets with logical partitions. The importance of having an admission control subsystem is accentuated with the support of both static and dynamic requests, mainly because the first is network intensive whereas the latter is CPU intensive. In a simple scenario the CPU could be causing a bottleneck when serving a high load of requests based on dynamic scripts, while the network and bandwidth are capable of serving static requests.

Right now the server only supports static HTML files. However, to be able to compete with Apache and other Web servers, AUC-Abyss needs to support dynamic scripts and CGI, which are commonly used nowadays. Therefore, one of the first possible future enhancements would be an add-on to support dynamic behavior. In addition, our Web server was built with C++ on Linux. Developing it in a standard programming language would make its portability easier, yet an operating system's dependency has to be architecturally addressed, and there are a lot of approaches that we can learn here from the development of portable operating systems to make AUC-Abyss portable across platforms. Other future plans for our server include adapting its architecture to provide distributed-based Web services and support for virtual servers [23].

The authors plan to release the Abyss server in the spring of 2005 under the GPL for research purposes only.

REFERENCES

[1] See http://apache.rcbowen.com/ApacheServer.html.

[2] V. Pai, P. Druschel, and W. Zwaenepoel, "Flash: An Efficient and Portable Web Server," *Proceedings of the 1999 USENIX Annual Technical Conference (Monterey, CA),* June 1999, pp. 199–212.

[3] Daniel A. Menascé, "Web Server Software Architectures," *IEEE Internet Computing*, vol. 7, 2003, pp. 78–81.

[4] See http://www.ele.uri.edu/Research/hpcl/Apache/journal_CA.pdf.

[5] E.D. Katz, M. Butler, and R. McGrath, "A Scalable Web Server: The NCSA Prototype," *Computer Networks and ISDN Systems*, vol. 27, no. 2, November 1994, pp. 155–164.

[6] A. Bestavros, R.L. Carter, M.E. Crovella, C.R. Cunha, A. Heddaya, and S.A. Mirdad, "Application-Level Document Caching in the Internet," *Proceedings of the 2nd International Workshop on Services in Distributed and Networked Environments (SDNE),* 1995.

[7] A. Luotonen and K. Altis, "World-Wide Web Proxies," *Proceedings of the First International Conference on the World-Wide Web,* 1994.

[8] M. Abrams, C.R. Standridge, G. Abdulla, S. Williams, and E.A. Fox, "Caching Proxies: Limitations and Potentials," *Proceedings of the 4th International Conference on the World-Wide Web (Boston, MA),* December 1995.

[9] C. Maltzahn, K.J. Richardson, and D. Grunwald, "Performance Issues of Enterprise Level Web Proxies," *Proceedings of the 1997 SIGMETRICS Conference on Measurement and Modeling of Computer Systems,* June 1997.

[10] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS),* December 1997.

[11] J. Gwertzman and M. Seltzer, "The Case for Geographical Pushcaching," *Proceedings of the 1995 Workshop on Hot Operating Systems,* 1995.

[12] S. Glassman, "A Caching Relay for the World Wide Web," *Proceedings of the First International Conference on the World-Wide Web,* 1994.

[13] A. Chankhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell, "A Hierarchical Internet Object Cache," *Proceedings of the 1996 USENIX Annual Technical Conference (San Diego, CA),* January 1996.

[14] A. Cockcroft, "Watching Your Web Server," http://www.sun.com/sunworldonline/swol-03-1996/swol-03-perf.html, March 1996.

[15] See http://www.microsoft.com/windowsserver2003/evaluation/overview/technologies/iis.mspx, April 2003.

[16] Brian Livingstone. "Intel Blows Bandwidth," http://itmanagement.earthWeb.com/columns/executive_tech/article.php/3068161, April 2003.

[17] See http://linuxtoday.com/news_story.php3?ltsn=2001-01-29-005-06-PR-HE-SV.

[18] See http://www.zeus.com/products/zws/features.html, Web Server Feature Comparison.

[19] See http://www.cs.princeton.edu/~vivek/flash/.

[20] See http://www.csse.monash.edu.au/~impp/Docs/Thesis%20Final.pdf.

[21] Thiemo Voigt and Per Gunningberg, "Adaptive Resource-Based Web Server Admission Control," *Proceedings of the 7th International Symposium on Computers and Communications,* 2002.

[22] Adam Bosworth, "Developing Web Services," *Proceedings of the 17th-International Conference on Data Engineering,* 2001.

[23] Valeria Cardellini, Emiliano Casalicchio, Michele Colajanni, and Philip S. Yu, "The State of the Art in Locally Distributed Web-Server Systems," *ACM Computing Surveys*, vol. 34, no. 2, June 2002.

BORIS LOZA

under attack

DEALING WITH MISSING UNIX FILES

Boris Loza is a founder of Tego System Inc. and HackerProof Technology, in addition to being a contributor to many industry magazines. He holds several patents and is an expert in computer security. He loves nature, reading books, and watching movies, and enjoys scuba diving and entomology.

bloza@hackerproofonline.com

A SECURITY BREACH CAN INSPIRE panic in administrators. This quick application note explains some techniques to be used to recover the names and contents of files during an attack or shortly thereafter.

Takedown, by Tsutomu Shimomura and John Markoff, describes the pursuit of Kevin Mitnick by Tsutomu Shimomura. It notes: "I could make out patterns of information still stored on my computer's disk that revealed the ghost of a file that had been created and then erased. Finding it was a little like examining a piece of paper on a yellow legal pad: even though the top page has been torn off, the impression of what was written on the missing sheet can be discerned on the remaining page."

In the UNIX and Linux worlds, just about everything is a file. UNIX treats regular files, directories, hard disks, printers, modems, and so on as files. When a file is created, it is assigned an inode (an index structure that is quicker for finding on-disk data structures than filename matching). When a file is deleted, the inode number is cleared from the directory, but the file does not vanish. The contents usually remain on the disk, at least for a while, until the disk blocks containing the contents are reused.

Listing Deleted Files

Because a directory is also a file, commands that manipulate files can be used to examine a directory. The od command performs an octal dump, which can include an ASCII listing. Let's consider an example of a directory (all outputs are taken from Solaris 9, except where otherwise indicated):

```
$ chdir testdir
$ ls -a
.       ..      Project     status      webstat.log
```

The ls command says the directory contains five entities: Project, status, webstat.log, an entry for the directory itself (.), and an entry for the parent directory (..). The on-disk structure can be examined using the od command (with a flag to display the ASCII characters, if the octal code is reasonable). Certain non-graphic characters appear as the intuitive C-language escapes; other non-printable characters appear as 3-digit octal numbers.

```
$ od -c .
0000000  \0  \b 023 337  \0  \f  \0 001    . \0  \0  \0   \0 013 006   P
0000020  \0  \f  \0 002   .  . \0  \0   \0  \b 023 340   \0 020  \0 007
0000040   P   r   o   j   e   c   t \0   \0  \b 023 341   \0 024  \0 013
0000060   w   e   b   s   t   a   t   .   l   o   g \0   \0  \b 023 342
0000100 001 304  \0 006   s   t   a   t   u   s \0  \0   \0  \0  \0  \0
0000120  \0  \0  \0  \0  \0  \0  \0  \0   \0  \0  \0  \0   \0  \0  \0  \0
*
0001000
$
```

The output starts each line with the number of bytes, expressed in octal, shown since the start of the file. The first line starts at byte 0. The second line starts at byte 20 (that's byte 16 in decimal, the way most of us count), and so on. One can easily see the listed file names on this output. It is easy to see that each file name is preceded by 8 bytes of some sort of file-system information. Additionally, file names are padded to the next 32-bit boundary (maybe after a \0 that terminates the file name).

Let's delete the status file and compare the od -c output with the previous one.

```
$ rm status
$ ls -a
.          ..         Project      webstat.log
$ od -c .
0000000  \0  \b 023 337   \0  \f  \0 001    . \0  \0  \0   \0 013 006   P
0000020  \0  \f  \0 002   .  . \0  \0   \0  \b 023 340   \0 020  \0 007
0000040   Project  \0   \0  \b 023 341  001 330  \0 013
0000060   webstat.log  \0   \0  \0  \0  \0
0000100 001 304  \0 006    status  \0  \0   \0  \0  \0  \0
0000120  \0  \0  \0  \0   \0  \0  \0  \0   \0  \0  \0  \0   \0  \0  \0  \0
*
0001000
```

Note that we still can see the status file, but four of the eight bytes that precede it (\b, \023, and \342) have been replaced by NULs, a zeroing of the inode number that links the name to the on-disk storage.

Understanding the Output

To understand more about all these outputs, let's take a look at the UNIX File System (UFS) directory structure that can be found in /usr/include/sys/ufs_fsdir.h:

```
struct  direct {
  uint32_t   d_ino;              /* inode number of entry */
  u_short    d_reclen;             /* length of this record */
  u_short    d_namlen;             /* length of string in d_name */
  char      d_name[MAXNAMELEN + 1]; /* name must be no longer than
this */
};
```

Now, we may better understand the output from od -c. We will analyze the initial output that was displayed above:

```
$ od -c .
0000000  \0  \b 023 337  \0  \f  \0 001    . \0  \0  \0  \0 013 006   P
0000020  \0  \f  \0 002   .  . \0  \0  \0  \b 023 340  \0 020  \0 007
0000040   Project  \0  \0  \b 023 341  \0 024  \0 013
0000060   webstat.log  \0  \0  \b 023 342
0000100 001 304  \0 006   status  \0  \0  \0  \0  \0  \0
0000120  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0001000
```

As you can see from this output, and based on the UFS directory structure, one can recognize the file names: d_name[MAXNAMELEN +1] (., .., Project, webstatus.log, and status). Before each file name, we can see a number that represents the length of string in d_name—001 for ., 002 for .., 007 for Project, 013 (11 decimal) for webstat.log, and 006 for status.

What can also be recognized in this output are two octal digits that represent the inode number in the list of inodes, d_ino. This is 013 006 for .., \b 023 337 for ., \b 023 340 for Project, and so on.

At this point, programmers might well think about writing a quick program to traverse the directory structure to print the files, maybe even flagging those with inode numbers of zero. Regrettably, the indexing information on some OSes is adjusted when a file is deleted. A better approach is simply to text-process the strings in the directory.

Probably the easiest way to do this is the strings command:

```
$ strings . > /tmp/a
$ cat /tmp/a
.
..
Project
webstat.log
status
```

It is quite possible to learn the names of deleted files (presuming the directory slots haven't already been reused). What about the data contained in deleting files?

Recovering Certain Text Files

In most UNIX and UNIX-like file systems, files are not necessarily recoverable after deletion. Sometimes, though, one can retrieve vital information.

LOG FILES

If you suspect that an intruder has modified or deleted your log file, you might try to recover what has been deleted from this file. When a file is deleted under the UNIX system, the inode number in the directory is set to 0, and disk blocks belonging to a file are marked "free" and returned to a pool of blocks that may be reused by the system. If the blocks of the deleted or altered file were not yet reused, this information is still present on the hard disk—somewhere.

Let's try to recover a log file. Assume that we log all network connections to /var/adm/messages with inetd -t. We suspect that an intruder modified our /var/adm/messages file to hide successful connections on June 4. Current output from /var/adm/messages:

```
$ tail messages
Jun  3 10:49:42 birch inetd[132]: exec[25727] from 10.56.49.194 2199
Jun  3 11:29:25 birch inetd[132]: exec[28958] from 10.56.49.194 2254
Jun  3 11:35:14 birch inetd[132]: telnet[29398] from 10.56.49.194 2255
Jun  3 14:04:21 birch inetd[132]: telnet[9711] from 10.56.53.55 57779
Jun  3 14:21:30 birch inetd[132]: ftp[10914] from 10.56.49.194 2430
Jun  3 14:51:04 birch inetd[132]: telnet[17225] from 10.56.49.194 2486
Jun  3 14:56:35 birch inetd[132]: telnet[17622] from 10.56.49.194 2487
Jun  5 09:55:00 birch inetd[132]: exec[14029] from 10.56.49.194 3248
Jun  5 11:13:33 birch inetd[132]: exec[19439] from 10.56.49.194 3281
Jun  6 14:17:14 birch inetd[132]: telnet[10520] from 10.56.49.194 3747
```

We can see that entries from Jun 4 are missing, a suspicious circumstance. Because, as we already know, almost everything in UNIX is a file, we can use the grep utility against the raw disk device on which /var/adm/messages is located for the string "Jun 4".

First, we must deduce which disk device holds (or held) the /var/adm/messages file (this output is produced on Solaris 2.6):

```
$ df /var/adm/messages
Filesystem      Kbytes    used     avail  capacity Mounted on
/dev/dsk/c0t0d0s4  290065    93826  167233 36%   /var
```

We will use grep against the entire /dev/dsk/c0t0d0s4 partition, all 2.9GB of it:

```
$ su -
Password:
Sun Microsystems Inc.  SunOS 5.6   Generic August 1997
# grep 'Jun  4' /dev/dsk/c0t0d0s4 > /tmp/123
# head /tmp/123
Jun  4 09:16:54 sundvl25 inetd[132]: telnet[2872] from 10.32.112.159
   1137
Jun  4 09:23:06 sundvl25 inetd[132]: telnet[3306] from 10.32.112.159
   1140
Jun  4 10:07:44 sundvl25 inetd[132]: ftp[6484] from 10.56.49.183 1072
Jun  4 10:08:17 sundvl25 inetd[132]: ftp[6519] from 10.56.49.183 1073
Jun  4 09:16:54 sundvl25 inetd[132]: telnet[2872] from 10.32.112.159
   1137
Jun  4 09:23:06 sundvl25 inetd[132]: telnet[3306] from 10.32.112.159
   1140
Jun  4 10:07:44 sundvl25 inetd[132]: ftp[6484] from 10.56.49.183 1072
Jun  4 10:08:17 sundvl25 inetd[132]: ftp[6519] from 10.56.49.183 1073
```

We redirected the output from the grep to the /tmp/123 file (choose any meaningless name in case the intruder is on the system), which is stored on a different partition of the hard disk. This will ensure that the data we are trying to recover is not overwritten and will avoid an embarrassing grep feedback loop.

Now, while the blocks that contain the file's contents might not be scanned in the proper order (there are no relevant rules about order of disk block allocation in a file system that has been in production for a while), we can see many of the entries that had been deleted from the /var/adm/messages file (based on the format of the messages file) by our intruder.

This process is time-consuming. Be patient. On a busy system it could take a lot of time to grep through the disk's blocks.

RECOVERING BINARY FILES

You may easily recover an executable file if it is still running as a process on your system. There may be situations in which a hacker runs the application and deletes an executable file. While the file's name is removed from a directory, the contents are still intact so that execution can proceed.

On Solaris and others, a link to the process exists in the /proc/[PID]/object/a.out directory. You may identify the process number for the deleted file by using the ps command or lsof utility.

For example, let's assume that we are going to restore a file that belongs to the process ID 22889 from the suspicious srg application that we found running on our system:

```
# ps -ef | more
UID   PID  PPID  C    STIME TTY      TIME CMD
root    0     0  0   May 10 ?       0:00 sched
root    1     0  0   May 10 ?       2:21 /etc/init -
...
root 22889 16318  0 10:09:25 pts/1    0:00 ./srg
root 16318 25824  0 08:56:27 pts/1    0:00 csh
```

As long as the command is still running, use cp to recreate its executable:

```
# cp /proc/2289/object/a.out /tmp/srg
```

The /proc directory connects to a pseudo-filesystem that abstracts the kernel's process architecture; /proc/2289/object/a.out is the process's executable binary.

You may also want take a look at Dan Farmer and Wietse Venema's The Coroner's Toolkit (TCT), which includes unrm and lazarus applications to automate this process.

Conclusion

Do not be frustrated if you cannot recover a deleted or altered file. Sometimes the odds of recovering a file are very slim. Be patient and take everything with a sense of humor!

STEVEN ALEXANDER

defeating compiler-level buffer overflow protection

Steven is a network test engineer at Front Porch in Sonora, CA. He gets to break things and shoot Nerf guns at people.

alexander.steven@sbcglobal.net

BUFFER OVERFLOW ATTACKS ARE THE most popular method intruders use to gain remote and privileged access to computer systems. Programs that fail to use appropriate bounds checking can allow an attacker to write data beyond the intended boundaries of a buffer and thus possibly corrupt control structures in the program. This enables an attacker to execute arbitrary code with the same privilege as the victim process. An attacker's preference is usually to overwrite the saved instruction pointer that is pushed onto the stack before a function call or to overwrite a function pointer that will be used later in the program.

It is also possible to use these attacks simply to overwrite other data. This kind of attack is harder to prevent but, fortunately, is less common than the previous type and is not discussed here.

Buffer overflows first gained attention with the release of the famed Morris worm which exploited a buffer overflow in fingerd [1]. Despite the attack used in the Morris worm, buffer overflows did not become popular until the release of two papers that detailed the discovery and exploitation of these vulnerabilities [2,3].

This paper discusses vulnerabilities in two compiler-level protection mechanisms, StackGuard and PointGuard. While this paper takes a critical look at both of these solutions, it does not intend to make them seem insignificant. The attacks described in this paper help to show how StackGuard and PointGuard should be complemented to construct a more complete protection system.

The reader should also note that PointGuard has not been publicly deployed. It was presented at the USENIX Security Symposium in 2003. The design might be changed before its release to correct functionality problems with some real-world software [4].

The reader should also note that StackGuard has reverted from the more advanced random XOR canary protection method analyzed here to the simpler terminator canary [5]. The justification for the change is that the attack method that prompted the change also enables an attacker to manipulate a program in ways that StackGuard cannot, and was not designed to, protect against. Because StackGuard has reverted to a weaker method and PointGuard is not available, the attacks in this paper are mostly of importance to the designers of new protection methods and have little consequence for currently deployed systems.

Exploiting a Buffer Overflow

To understand how a buffer overflow exploit works, we must first understand how a function call occurs:

1. The calling procedure pushes any function arguments onto the stack in reverse order.
2. The calling procedure executes a "call" instruction, which pushes the address of the next sequential instruction onto the stack and tells the processor to transfer execution to the target function.
3. Assuming that frame pointers are being used, the called function pushes the old frame pointer onto the stack and copies the value stored in the stack pointer over the frame pointer. Then, the stack pointer is decremented (the stack grows down) to make room for local variables.

Figure 1 show the stack layout for a called function with a single variable (a character array).

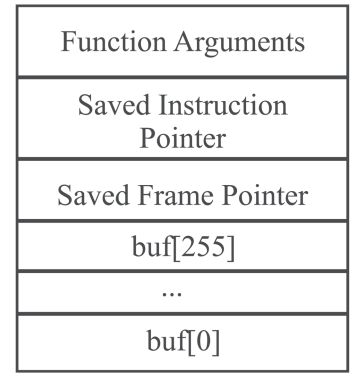


FIGURE 1: STACK LAYOUT FOR A CALLED FUNCTION WITH CHARACTER ARRAY ARGUMENT

The function epilogue consists of popping the saved frame pointer from the stack and executing a return instruction. The return instruction causes the processor to pop the saved instruction pointer from the stack into the program counter and begin execution at that address. The saved instruction pointer is supposed to hold the instruction address that was saved on the stack in step 2 above.

Consider the following code:

```
#include <stdio.h>

int main (int argc, char *argv[]) {
    char buf[256];
    if(argc < 2) {
       printf("Oops.\n");
       return -1;
    }
    strcpy(buf, argv[1]);
    return 0;
}
```

This snippet of code is vulnerable to a trivial buffer overflow attack. The strcpy function does not perform bounds checking (unlike its cousin strncpy), so the program will copy characters from argv[1] to buf until the program crashes or strcpy encounters a null character, \0. An attacker could find a way to provide a carefully crafted input that will cause this function to execute his own code instead.

First, such an attacker would assemble a small bit of code that will do something useful such as the semantic equivalent of exec ("/bin/sh"). Such code is usually

referred to as "shellcode" since the popular use is to execute a command shell. Shellcode can be used to do more complicated things, such as open a network connection or add a new root user. There are some restrictions as to how this code can be constructed. For instance, there cannot be any null characters in the resulting machine code. Aleph One discusses constructing workable shellcode [3]. There is quite a lot of shellcode available online so, unfortunately, aspiring exploit writers don't have to start from scratch.

In order to execute some shellcode, an attacker provides the code as a part of the input to a vulnerable program. The attacker crafts the input so that it will exceed the bounds of the allocated buffer and overwrite the saved instruction pointer with the address of the provided shellcode. If the attacker does not know the exact address at which the shellcode will be stored, he can prepend a series of null instructions (NOPs) to the shellcode. If the provided address points to any location within the series of NOPs, execution will continue through the NOPs and eventually reach the shellcode. If the attacker does not know the exact location of the saved instruction pointer (common if the attacker doesn't have access to the source code), he may duplicate the shellcode address several times. In such a case, it might take the attacker a few tries to overwrite the saved instruction pointer on the correct 4-byte boundary.

It might also take the attacker a few extra tries to guess the correct shellcode address. The address at which the shellcode is stored is usually not difficult to guess, even in black-box analysis, since the stack begins at a known location. This does not hold true if the target program runs on a system with good stack randomization. Figure 2 shows the attacker's input layout. Figure 3 depicts the manner in which this input corresponds to the function stack layout.

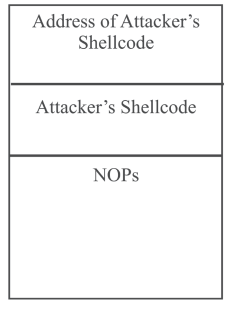


FIGURE 2: ATTACKER'S STACK LAYOUT AFTER INPUT

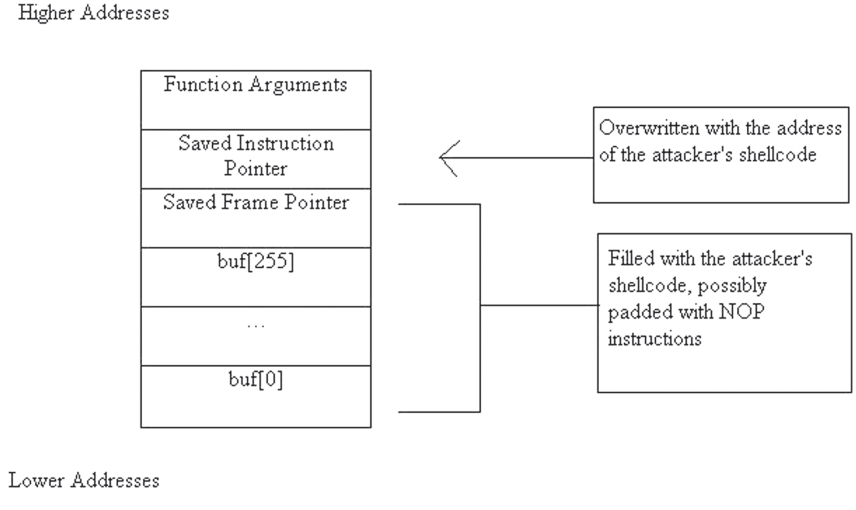


FIGURE 3: EXPLICATION OF FUNCTION STACK LAYOUT WITH ATTACK

Solutions, a Survey

Many methods have been proposed to prevent the execution of buffer overflow attacks [6], some of which are discussed here. Papers about several solutions and attacks are available on Purdue University's SmashGuard buffer overflow prevention page [7].

OS-LEVEL

NON-EXECUTABLE STACK

One of the first methods, the non-executable stack, was proposed by Solar Designer [8]. A non-executable stack prevents the standard buffer overflow attack which modifies the saved instruction pointer so that it points at the attacker's shellcode. The attacker's shellcode is normally stored in the same stack-allocated buffer that was overrun to change the instruction pointer. If the stack is non-executable, the attempt to resume execution at this location will fail.

This defense can be defeated by injecting executable code into other data areas, such as the standard .data and .bss sections. The defense was also defeated by Solar Designer [9] and Rafal Wojtczuk [10] using the return-into-libc method. In this method, the saved instruction pointer is modified so that the program will return into an instruction sequence in the C library. It is not necessary that the instruction pointer direct execution to the beginning of a function in the C library. Often, an attacker will wish to point at a call to system() inside one of the C library functions. An attacker can manipulate the stack so that his provided arguments will be used in the call to system().

PAX/ASLR

Randomizing the base address at which libraries are loaded can hinder return-into-libc attacks (used to defeat non-executable restrictions such as those in Solar Designer's stack patch). This technique was introduced in [11] and used by ASLR in PaX [12]. In early versions of PaX, an attacker could defeat this by instead returning into the Process Linkage Table (PLT) [13,14]. The PLT is used to resolve libc (and other) function addresses automatically. Currently, PaX can also randomize the executable base for ELF executables [12]; this prevents the return-into-PLT attack. There is another attack that can be used against PaX/ASLR with the randomized executable base in effect [15]. The attack uses a partial overwrite of the saved instruction pointer to gain control over the arguments passed to printf, which allows an attacker to discover information about the randomized library base using a format string attack so that a normal return-into-libc attack can be performed. When used with PaX, StackGuard and ProPolice/SSP can both prevent these attacks. The OpenBSD project has implemented W^X, which uses techniques similar to PaX. OpenBSD also uses address randomization and ProPolice/SSP.

COMPILER-LEVEL

STACKGUARD AND PROPOLICE/SSP

Another possible solution was proposed by Crispin Cowan and is used in StackGuard [16,5]. StackGuard places a canary value between the saved frame and instruction pointers and the local function arguments. Figure 4 shows the revised stack layout. The canary value is set in the prologue to each function and is checked for validity in the epilogue. If the canary value has been modified, a

handler function is called and the program terminates. A direct attack will overwrite the canary value before it overwrites the saved instruction or frame pointers. Any of three types of canary can be used: a terminator canary, a random canary, or a random XOR canary.

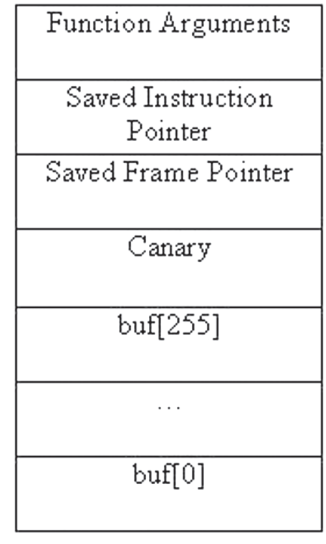


FIGURE 4: REVISED STACK LAYOUT WITH CANARY

A terminator canary contains multiple terminator values, such as a NULL byte or newline, which are used to indicate the end of a string in the various C library string functions. Because these values are used to terminate a string, an attacker cannot avoid changing them with a direct buffer overrun. It is possible to repair a terminator canary if an attacker has the opportunity to perform multiple overruns in one function. The first overrun can be used to change the instruction pointer and the subsequent overrun can be used to repair the canary by lining up the terminator in the string with the corresponding value in the terminator canary.

A random canary is a random value chosen at runtime. The random value is stored in a global variable and is used for each function in a program. It is stored in the same manner as the terminator canary. It is assumed that an attacker will be unable to overwrite the global value or to cause the program to leak the value. In some circumstances it is possible, however, to force the program to leak the random value using a format string attack. Overwriting the global variable is not useful since an attacker could just as easily overwrite a function pointer (.got entry, .dtors, etc.).

The random XOR canary was introduced into StackGuard to prevent an attack published in *Phrack Magazine* [17]. Rather than directly overwriting the canary and saved instruction pointers, an attacker can overwrite a data pointer that will be used later in the function as the destination for a string or memory copy that uses attacker-supplied data. The attacker can modify the pointer so that it points directly at the saved instruction pointer. When the attacker's data is copied to that address later in the function, the saved instruction pointer will be overwritten without modifying the canary.

With the random XOR canary, a random value is again generated at runtime and stored in a global variable. Rather than storing the random value on the stack, the random value is XORed with the saved instruction pointer and the result is stored on the stack. During the function epilogue, the saved canary is XORed with the random value and the result is compared to the saved instruction

pointer. If the values do not match, the handler function is called and the program terminates. The maintainers of StackGuard have reverted to using the terminator canary because the attack used to defeat the terminator canary can also be used to corrupt other important values such as function pointers.

SSP, previously known as ProPolice, is based on StackGuard and uses a random canary [18]. SSP offers several improvements over StackGuard, however, and is more difficult to defeat. SSP reorders local function variables so that pointers are stored below buffers in memory (i.e., higher on the stack). This rearrangement prevents an attacker from successfully employing attacks such as the one used to defeat StackGuard. There is a limitation to this: the variables within a data structure cannot be reordered, so it is possible for an attacker to exploit a buffer overflow within a data structure and overwrite a pointer value within that same structure. This does not seem (to me) to be a common problem.

SSP also copies function arguments to the local stack frame. An attacker can target the arguments of a function if they will be used inside the function after he modifies them. In some cases, an attacker can use them (perhaps by overwriting a pointer value) to write arbitrary data to any writable location in memory. The canary value will be overwritten but, since an attacker can write anywhere, he could also overwrite the address in .got of one of the functions used in the handler function that is called to terminate the program. By copying the function arguments to a local memory area below the local variables, SSP prevents this.

StackGuard and SSP cannot prevent attacks that occur in heap memory [19,20,21]. Early versions of StackGuard did not attempt to protect the saved frame pointer. If the frame pointer is not protected, StackGuard can be bypassed by taking control of the stack frame [22].

PointGuard

PointGuard protects pointer values inside programs, a technique that promises much better protection than using StackGuard alone [23]. PointGuard works by XOR-encrypting pointer values with a random value determined at runtime and stored in a global variable. Code is added to a protected program to decrypt pointer values automatically before each use. Pointer values are decrypted only in registers, and the decrypted pointer is not stored in memory. Without knowledge of the random value used to encrypt the pointers in a program, an attacker cannot overwrite a pointer and hope for a meaningful decryption. If an attacker overwrites a pointer hoping to point to an exact location, his chances are 1 in 2^{32}, or about 1 in 4 billion. An attacker has a much better chance if he is trying to point a function pointer at NOP-padded shellcode, but even with a 1-kilobyte NOP buffer, his chances are only about 1 in 4 million. Dereferencing a random pointer value is likely to cause a segmentation violation, which will cause the targeted program to exit and dump core.

Unlike StackGuard and SSP, PointGuard does provide protection against heap attacks. Note that in order to provide protection against malloc and free attacks, libc must be compiled with PointGuard. Unfortunately, PointGuard can be defeated using format string attacks, as discussed on Bugtraq [24] and using an attack detailed below. An implementation of PointGuard has not been publicly released.

Format String Vulnerabilities

Format string vulnerabilities arise when functions that accept format strings and a variable number of arguments (e.g., printf) are used without a programmer-

provided format string. If the function is used to process user-provided input, a malicious user can supply his own format string. An attacker can use specially crafted input to leak information from the victim program (most likely by walking the stack) or to overwrite arbitrary data. (For an introduction to format string exploits, see [25]; for more advanced techniques, see [26] and [27].)

In the printf family of functions, data can be overwritten using the %n format specifier. The %n specifier stores the number of bytes that printf has written so far at the provided address. An attacker can use this feature to overwrite a pointer (including a function pointer), a saved instruction address, an entry in the Global Offset Table (GOT) [14], or any other value in memory that can be changed to aid an attacker in diverting a program's execution or elevating privilege.

While some RISC systems have alignment requirements for writes that use the %n specifier, Intel-based systems do not. Because of this, the %n specifier can be used multiple times, with each write operation targeting an address just one byte higher than the previous operation. In this case, only the least significant byte of each count is used to construct a new value for a 32-bit word. This technique has the consequence that it will also overwrite three bytes adjacent to the target value. This is usually not a problem for an attacker. If, for instance, an attacker uses this method to overwrite a saved instruction pointer, the first three bytes lower in the stack (at a higher memory address) will be corrupted. Normally, this value will be one of the arguments passed in to the current function. If this value is dereferenced after the attacker corrupts it, the program may crash. If, on the other hand, it is not, the attacker can cause the program to execute arbitrary code when it exits from the current function. If this is a problem, the attacker need only overwrite another value, such as _atexit or a GOT entry, instead.

Attacker-provided format strings can also be used to leak information from the currently running program. The %iii$ specifier is extremely useful in this regard. In this specifier, iii is the number of the argument to print; for instance, %2$08x will print the second argument on the stack in zero-padded hexadecimal format. This can be used to "walk" the stack or to print arbitrary values directly. This technique was crucial in gathering information for the return-into-libc exploit used to defeat PaX [15]. In that particular case, the least significant byte of the saved instruction pointer was overwritten by a buffer overflow to cause a vulnerable function to return directly to a printf call in the middle of that same function. In doing this, the author was able to cause his own arguments to be provided to the printf function instead of those that were hard-coded into the program. The author used this technique to force the program to leak the information necessary to execute a return-into-libc exploit on a PaX protected system with ASLR. The target function was not otherwise vulnerable to a format string attack. The format string attack was made possible only by the buffer overflow, which prevented the correct values from being placed on the stack before printf was called.

A New Weakness in PointGuard

In addition to the previously discussed vulnerability to information leaking with format strings, PointGuard is also vulnerable to buffer overflows and to data manipulation with format strings. The claim given in the PointGuard paper [23] is that an attacker can destroy a pointer value but cannot produce a predictable pointer value. This is not completely true.

PointGuard is weak because pointer encryption is achieved by using a bitwise exclusive-OR operation rather than a more complex nonlinear operation. Because of this, any byte of the encrypted pointer that is not overwritten will

still decrypt correctly. This enables an attacker to make use of partially overwriting a pointer. If an attacker can find a situation in which it is advantageous to redirect a pointer toward a location whose most significant one to three bytes are the same as the location that the pointer originally referenced, he can, by brute force, attempt to redirect the pointer to this new location with far less effort than would be required to brute-force a 32-bit value.

On little-endian architectures, an attacker can use a simple buffer overflow to overwrite the least significant bytes of a pointer value, since the least significant bytes are stored at a lower address and thus overwritten first. Using format string attacks, which allow considerable flexibility in the way a value is overwritten, an attacker can bypass PointGuard on both little-endian and big-endian systems.

Consider the following code, a variation of the vulnerable "straw man" program included in the PointGuard paper:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define ERROR -1
#define BUFSIZE 64

int goodfunc(const char *string) {
    printf("%s\n", string);
    return 0;
}

int main(int argc, char **argv) {
    static char buf[BUFSIZE];
    static int (*funcptr)(const char *str);
    if(argc <= 2) {
        fprintf(stderr, "Usage: %s <buf> <goodfunc arg>\n",
        argv[0]);
        exit(ERROR);

    }
    funcptr = (int (*)(const char *str))goodfunc;
    memset(buf, 0, sizeof(buf));
    strncpy(buf, argv[1], strlen(argv[1]));
    (void)(*funcptr)(argv[2]);
    return 0;
}
```

I compiled this code on an Athlon XP running FreeBSD 4.9. When the executable is loaded, the goodfunc function is located at 0x080485c4 and the buf buffer is located at 0x08049940. An attacker who loaded buf with his own executable shellcode would only need to overwrite the two least significant bytes of funcptr correctly in order to execute his code instead of goodfunc. Since those two bytes are XOR-encrypted with 16 random bits, an attacker who overwrites the two least significant bytes of funcptr will have a 1 in 65,536 chance of redirecting that pointer to the beginning of his shellcode. While this might be difficult to accomplish remotely before the attack is noticed by a system administrator, such an attack could be accomplished locally without any trouble.

Obviously, this example is contrived and does not necessarily provide a realistic memory layout for a real-life program. Instead, let us consider the layout information (see Figure 5, below) from three real, privileged programs from FreeBSD 4.9: lpr, ftpd, and rcp. In the case of lpr, redirecting a vulnerable pointer in a PointGuard-protected instance of the program would require an amount of effort similar to our example, since the .text and both data sections are located inside the same 16-bit segment. The other two programs would be more difficult to subvert since the data sections and the .text section share only the most significant eight bits of their addresses. An attacker would thus be required to over-

write the lower 24 bits of a function pointer in order to redirect it to his injected shellcode in one of these sections.

The odds of an attacker providing a 24-bit value that will correctly decrypt to the address of his shellcode are slightly better than 1 in 17 million. The outlook for an attacker is not quite so bleak, however. If the attacker is able to place shellcode in more than one location or to prepend a long series of null operations (NOP) to his shellcode, he can increase his odds tremendously.

Assume that an attacker's shellcode is only 50 bytes (a number well within the normal range). Further, assume that he is able to place this shellcode at the end of a one-kilobyte buffer after padding the buffer with NOPs. The attacker's odds increase to one in 17,000. In some situations, the attacker may be able to construct an even longer series of NOPs by having access to a large character array or by overwriting several data structures with the NOPs and shellcode without that data being molested before the altered function pointer is dereferenced. In highly favorable situations, an attacker might be able to guess a correct value with only a few thousand guesses on average. Clearly, such situations do not correspond with the argument in the PointGuard paper that an attacker cannot meaningfully corrupt a pointer without knowledge of the PointGuard encryption key.

In general, the complexity of guessing a value that will successfully cause a function pointer to reference NOP-padded shellcode is $2^{(X-\ln(\textit{number of NOPs}))}$ where X is the number of bits guessed.

On little-endian systems, the security of PointGuard can be improved slightly by rotating a pointer value one byte to the left after the XOR encryption and rotating it back before the XOR decryption. In most situations, this would force an attacker to overwrite the entire 32-bit value. Using format string attacks, it would still be possible in some circumstances to overwrite only the least significant three bytes. Still, such situations are likely to be far more rare than those in which an attacker can corrupt a pointer with a simple buffer overflow. Unfortunately, such a change is likely to at least double the current performance penalty imposed by PointGuard.

Program	.text	.data	.bss
/usr/bin/lpr	0x0804964c	0x0804f140	0x0804f400
/usr/libexec/ftpd	0x0804a974	0x08059ce0	0x0805a560
/bin/rcp	0x080480b8	0x08081ae0	0x080831a0

FIGURE 5: ACTUAL MEMORY LAYOUTS FOR THREE COMMON PROGRAMS

A New Weakness in StackGuard

In this section, all references to StackGuard should be interpreted to mean StackGuard with the random XOR canary [5,17].

StackGuard has a weakness that corresponds to the previously discussed vulnerability in PointGuard. The random XOR canary is the result of exclusive-ORing a random canary value (generated at runtime) with the saved instruction pointer. The result is stored on the stack after the saved instruction and frame pointers and before the local function variables. Code in the function epilogue exclusive-ORs the saved canary with the random value (thus canceling out the effect of the random value) and compares the result to the saved instruction pointer. If the two values do not match, the program exits.

Since exclusive-OR is a bitwise operation, if only some bytes of the saved instruction pointer are modified, then only the corresponding bytes of the saved canary value need to be modified. The bytes of the saved canary can be overwritten with any random value.

This weakness is more difficult to exploit than the one in PointGuard. The conditions that must exist in a program's code for exploitation to be possible are more specific. The value used to overwrite the saved canary must be equal to the result of exclusive-ORing the pertinent bytes of the random canary and the attacker-supplied instruction pointer value, since a direct comparison is used in the function epilogue to determine whether the exclusive-OR of the random canary and the saved instruction pointer match the saved canary. It is still possible to overwrite the saved canary value with any random or fixed value, since the random canary used by the program changes with each execution.

PointGuard offers more room for error because it does not perform a direct comparison; instead, PointGuard allows the pointer to be dereferenced, under the assumption that a corrupted pointer will decrypt to a random value and most likely reference an invalid memory region, which will cause the program to crash. With PointGuard, an attacker can inject NOP-padded shellcode, which allows him the opportunity to guess a value that will decrypt to any location within the series of NOPs (or the first useful instruction in the shellcode).

If a format string overwrite attack is used to circumvent StackGuard, the attack is fairly straightforward. The attacker uses the `%n` modifier to overwrite all or part of the saved instruction pointer with a newly constructed value of his choosing. The attacker also uses the `%n` modifier to overwrite the corresponding bytes of the saved canary with any random or fixed value (probably fixed).

If an attacker overwrites the entire saved instruction pointer, he must also overwrite the entire saved canary. In this case, his attack has less than a 1 in 4 billion chance of success. An attacker's goal will be to find a situation in which he is able to inject code at a location that shares one or two significant bytes with the value of the original saved instruction pointer (as in the above PointGuard attack).

In order to bypass StackGuard using traditional techniques, an attacker must use a buffer overflow to overwrite the least significant bytes of the saved canary value. The attacker can overwrite these bytes with any value, fixed or random. He must also overwrite a data pointer so that it points directly at the saved instruction pointer. This modified data pointer must later be used as the destination for a string or memory copy that uses user-supplied input. An attacker will use the string or memory copy to point the saved instruction pointer at his shellcode (or to perform a return-into-libc attack). The affected data pointer must point directly at the saved instruction pointer; there is no margin for error as when attempting to point at NOP-padded shellcode.

Assuming that an attacker is successful in overwriting a pointer value and that he uses the corrupted pointer to correctly overwrite the saved instruction pointer, this attack will fail in each instance that the saved canary value is not equal to the exclusive-OR of the random canary (generated at each execution of the program) and the attacker-supplied return address. Since the random canary changes with each execution of the program, an attacker can supply any fixed or random value to overwrite the least significant bytes of the saved canary and will eventually succeed.

Consider the following source code:

```
int main(int argc, char **argv) {
    char *ptr;
    char buf[256];
    ...
    strcpy(buf, argv[1]);
    do_some_parsing(buf);
    strcpy(ptr, buf);
}
```

This program is vulnerable to a standard buffer overflow attack; an attacker can provide input that will be copied beyond the boundaries of the buf array, potentially overwriting the saved frame or instruction pointers that are stored on the stack between the function arguments and the local variables.

StackGuard will prevent a generic buffer overflow attack against this code. If an attacker attempts a standard buffer overflow attack against the saved instruction pointer, the canary value will be overwritten, StackGuard will detect the modification in the function epilogue, and the attack will fail.

In the versions of StackGuard that use a random or terminator canary, the previously published attack [17] applies and ptr can be overwritten instead. Instead of attacking the saved instruction pointer, an attacker can use a stack overflow to modify ptr so that it points at the return instruction pointer. The second strcpy operation will then overwrite the instruction pointer with the contents of buf without modifying the canary.

Although the random XOR canary prevents a direct application of this attack, the attack remains possible with some modifications. An attacker can still use ptr to overwrite part or all of the saved instruction pointer. In addition, he will have to overwrite the bytes of the canary that correspond to the bytes of the instruction pointer that he modifies. If he modifies every byte of the saved instruction pointer, his chances of success are slim, because he will have to overwrite the entire canary and will have less than a 1 in 4 billion chance that he will overwrite the canary with the correct value.

To improve his chances, the attacker will have to inject code into the .bss or .data memory regions, which often share the one or two most significant bytes of their addresses with the .text section. Alternatively, the attacker can attempt a return-into-PLT attack, since the .plt section often shares the most significant bytes of its address with the .text section. By using a return-into-PLT attack or injecting code in the .bss or .data sections, an attacker can redirect control of the program by only partially overwriting the saved instruction pointer and, consequently, only partially overwriting the saved canary.

This attack is an extension of the technique used to bypass StackGuard [17]. In the *Phrack* article, a string pointer was overwritten using a simple stack overflow. The pointer was later used as a destination pointer for a string copy which overwrote the saved instruction pointer with the location of either attacker-supplied shellcode or the address of a libc function (for a return-into-libc attack). The attack in this paper carries the restrictions that the saved instruction pointer should only be partially overwritten in order to ensure a reasonable chance of success and that corresponding bytes of the saved canary value must also be overwritten.

Conclusion

The attack against StackGuard is easy to ameliorate since it depends on exact knowledge of the location of the saved instruction pointer on the stack. Runtime and load-time stack randomization [11] greatly increase the difficulty of this attack. Load-time stack randomization can be implemented with only a few lines of code on most operating systems [28]. The difficulty of this attack is multiplied by the amount of stack randomization applied. Thus, if an attacker constructs an exploit that has a 1 in 16 million chance of success (he modified three bytes) and the attack is used against a system that uses 10 bytes of stack randomization, the chance of success drops to less than 1 in 16 billion. PaX uses 24 bits of stack randomization; the code published in *;login:* uses 18. Load-time stack randomization carries a negligible performance penalty at load-time and does not affect runtime performance at all.

The attack against StackGuard is not possible when PointGuard is used. Under most circumstances, this attack would not be possible if StackGuard used local variable reordering as in ProPolice/SSP.

The use of buffer overflows against PointGuard is possible only under specific circumstances. SSP's local variable reordering has no runtime performance penalty and would make these circumstances extremely rare. FormatGuard can likewise protect PointGuard against most format string attacks [29]. Unfortunately, FormatGuard protects only calls to the C library. Programs such as wu-ftpd, which use an alternative implementation of printf, would not be protected. In some circumstances, the combination of SSP, PointGuard, and FormatGuard would still be vulnerable. Replacing SSP with StackGuard makes the combination even weaker.

The non-executable restrictions imposed by PaX and W^X would make the attack against PointGuard difficult because an attacker would not be able to execute code injected into the .data or .bss sections. The various memory randomization features of ASLR would make it even more difficult for an attacker to meaningfully redirect a pointer value.

The use of compiler-level stack protection, as in StackGuard and SSP, along with PaX, can defeat the attacks that have been published for defeating PaX alone. Some more advanced variations on these attacks may be possible, but the pointer protection offered by SSP's local variable reordering is likely to prevent most of them. Even without the benefit of StackGuard or SSP, the attacks against PaX are more difficult than the above attack against PointGuard.

Pointer encryption, canary protection methods, and execution restriction mechanisms have all been shown to be vulnerable to various attacks. The risk of a successful attack against these systems can be reduced if a host intrusion detection mechanism such as Segvguard [13] is used to prevent a program from executing after some number of crashes. A mechanism such as Segvguard is necessary to complement PointGuard, PaX, W^X, or any address space randomization.

REFERENCES

[1] Eugene Spafford, "The Internet Worm Program: Analysis," *Computer Communications Review* (January 1989).

[2] Mudge, "How to Write Buffer Overflows" (October 1995) http://www.insecure.org/stf/mudge_buffer_overflow_tutorial.html.

[3] Aleph One, "Smashing the Stack for Fun and Profit," *Phrack Magazine* 49 (November 1996), http://www.phrack.org/49/P49-14.

[4] Crispin Cowan, personal communication, January 2004.

[5] Crispin Cowan and Perry Wagle, "StackGuard: Simple Stack Smash Protection for GCC," *Proceedings of the GCC Developers Summit* (May 2003).

[6] Crispin Cowan et al., "Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade," *DARPA Information Survivability Conference and Expo (DISCEX),* January 2000.

[7] See http://engineering.purdue.edu/ResearchGroups/SmashGuard/.

[8] Solar Designer, "Non-Executable User Stack," http://www.openwall.com/linux/.

[9] Solar Designer, "Getting Around Non-Executable Stack (and Fix)." http://www.securityfocus.com/archive/1/7480.

[10] Rafal Wojtczuk, "Defeating Solar Designer's Non-Executable Stack Patch" (January 1998), http://www.securityfocus.com/archive/1/8470.

[11] Monica Chew and Dawn Song, "Mitigating Buffer Overflows by Operating System Randomization," *Tech Report CMU-CS-02-197* (December 2002).

[12] See http://pax.grsecurity.net/docs/index.html.

[13] Nergal, "The Advanced return-into-lib(c) Exploits: PaX Case Study," *Phrack Magazine* 58 (December 2001), http://www.phrack.org/phrack/58/p58-0x04.

[14] John R. Levine, *Linkers and Loaders* (San Diego: Academic Press, 2000).

[15] Anonymous, "Bypassing PaX ASLR Protection," *Phrack Magazine* 59 (July 2002), http://www.prhack.org/phrack/59/p59-0x09.

[16] Crispin Cowan et al., "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks," *7th USENIX Security Symposium* (January 1998), pp. 63–77.

[17] Bulba and Kil3r, "Bypassing StackGuard and StackShield," *Phrack Magazine* 56 (May 2000), http://www.phrack.org/phrack/56/p56-0x05.

[18] Hiroaki Etoh, "ProPolice: GCC Extension for Protecting Applications from Stack-Smashing Attacks," IBM (April 2003), http://www.trl.ibm.com/projects/security/ssp/.

[19] Matt Conover, "w00w00 on Heap Overflows" (January 1999), http://www.w00w00.org/files/articles/heaptut.txt.

[20] Michel Kaempf, "Vudo Malloc Tricks," *Phrack Magazine* 57 (August 2001), http://www.phrack.org/phrack/57/p57-0x0b.

[21] Anonymous, "Once upon a free()," *Phrack Magazine* 57 (August 2001), http://www.phrack.org/phrack/57/p57-0x0c.

[22] Gerardo Richarte, "Bypassing the StackShield and StackGuard Protection" (April 2002), http://www1.corest.com/corelabs/papers/index.php.

[23] Crispin Cowan et al., "PointGuard: Protecting Pointers from Buffer Overflow Vulnerabilities," *12th USENIX Security Symposium* (August 2003), pp. 91–104.

[24] Crispin Cowan, "Re: PointGuard: It's not the Size of the Buffer, it's the Address." http://www.securityfocus.com/archive/1/333988.

[25] Pascal Bouchareine, "Format String Vulnerability" (July 2000), http://www.hert.org/papers/format.html.

[26] scut and Team Teso, "Exploiting Format String Vulnerabilities" (September 2001), http://www.team-teso.net/articles/formatstring/.

[27] gera and riq, "Advances in Format String Exploitation," *Phrack Magazine* 59 (July 2002), http://www.phrack.org/phrack/59/p59-0x12.

[28] Steven Alexander, "Improving Security with Homebrew System Modifications," *;login:*, vol. 29, no. 6 (December 2004), pp. 26–32.

[29] Crispin Cowan et al., "FormatGuard: Automatic Protection from printf Format String Vulnerabilities," *10th USENIX Security Symposium* (August 2001).

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@usenix.org*

IT'S A BLUSTERY SPRING DAY AS I write this column. I've been reading a great new book about Internet Denial of Service (see details in [1]), and I have to confess I am depressed by what I am reading. The outlook just isn't very good, for so many reasons.

To top things off, some people in the German branch of the Honeynet Project have published a new paper about their experience collecting bots [2]. Their paper describes some of the bots captured, as well as information about the use of IRC for command and control (C&C). The sizes of botnets uncovered were not that enormous (well, just tens of thousands in some cases), but as the paper points out, a botnet of 1000 agents, with an average Internet connection speed of 128Kbps, can easily swamp a target with a 100Mbps connection to the Internet.

I haven't really studied the newer distributed denial of service (DDoS) agent software much in recent years, not since the beginning of the century. Back in late 1999, people were worried about big DDoS attacks being used to take out large parts of the Internet. When New Year's Eve passed without incident, people breathed a sigh of relief. But that relief was short-lived, as heavily publicized attacks on commercial Web sites in February 2000 showed.

The early DDoS tools, like Trinoo and Tribe Flood Network, have been replaced by newer, much more flexible tools. Chief among these are descendants of Agobot [3], a bot written in excellent C++ that can be compiled for either Windows or Linux. Agobot uses IRC channels for communications, unlike the earlier DDoS agents that relied on receiving commands from handlers. The commands used with older DDoS agents had easily recognizable signatures that were soon included in IDS software. Researchers also wrote tools that could probe for DDoS agents. But the use of IRC implies that any network that permits outgoing IRC connections will also permit DDoS agents to receive commands and report to the person running the botnet.

I do need to step back for a moment and define my terms. In the world of Internet Relay Chat (IRC), bots, short for robots, were originally network programs that would stay connected to an IRC server and thus stay active in a particular channel. The bot would perform services for its owner—for example, bestowing special privileges that the owner would lose if he left the channel. Bots run on systems other than the bot owner's, so that a denial of service attack against the bot owner's system would leave the bot still running [4].

Over time, bots gained additional abilities. Those who frequent IRC channels like #hack often fight over control of the channel, and one proven method of knocking someone out of a channel is to use DDoS attacks. A person who can amass a large network of bots (a botnet) can use this network to flood any adversary's Internet connection at will.

But why stop there? The Honeynet paper goes on to describe the many other features of advanced bots. They include the ability to execute any command, update their own software, hide themselves, sniff the network, log keystrokes, launch worms (like the Witty worm), and relay spam. Bots can be used to steal identity information, as well as license information for games and software (Agobot is big on this). Agobot, when running on Windows systems, also attempts to disable firewall and anti-virus software.

Botnets have been used for other financial purposes than simple identity theft. People have used them as a blackmail threat, one that can easily be demonstrated by launching a short flood against the target. Some botnet owners will hire out their botnets for DDoS attacks, or to spammers for relaying email through the use of SOCKS (the proxy server by David Koblas announced during the 1992 USENIX Security Symposium).

The German Honeynet Project paper collected information about bots by setting up several GenII [5] honeynets with Windows victims. They had decided on Windows systems as targets based on the amount of scanning detected for Windows-specific services. In their paper, the authors claim that Windows XP (SP1) and Windows 2000 were, by far, the most popular hosts for bots, followed by other Windows versions. They would collect software installed on the honeypot and reinstall the OS each day. On average, the Windows box would be owned in 10 minutes. In one instance, a box was compromised within three seconds after being connected to the network.

The group later designed a special program, mwcollect2, that simulated vulnerabilities and would download malware when commanded to do so by the exploit. This tool made it much easier to collect bots and other malware.

Over four months of research, the German Honeynet group tracked over 100 botnets that used IRC for C&C. Through the use of IRC JOIN messages, they saw 226,585 unique IP addresses of bots connecting to the IRC channels the group was tracking. This number is deceptive, in that there is no way of knowing if the bot's host was assigned a different IP address over time by DHCP. Also, many of the IRC servers used, especially by the more sophisticated botnet owners, were hacked so that they would not provide JOIN messages or accept commands that could list the IP addresses of participants. But the researchers estimate that at least one million hosts have bots installed.

Defenses

The Honeynet Project paper discusses the attackers. The Internet DoS book talks about attack tools, but focuses on defensive techniques. Imagine that you want to defend your site against DDoS attacks? What will you do if the attacker wants to send, on average, 1Gbps of reasonable seeming traffic at your network. Remember that this volume requires 10,000 bots and does not appear to be impossible for some attackers. The German researchers monitored one botnet with 50,000 hosts.

If you look at the information about the duration of DDoS attacks [2,6], most, but not all, attacks are short-lived. Most bots and agents accept time periods measured in seconds, with many attacks lasting only minutes. But some attacks can go on for weeks. I find this distressing to consider.

Chapter 5 of the DoS book includes an excellent discussion of where to locate DDoS defenses. While it would be ideal to filter out attacks at the source, for example, you will quickly realize, as the authors point out, that you do not control ISPs. If edge networks and ISPs would, at the very minimum, enforce ingress filtering by permitting only non-spoofed source addresses from their clients' systems, we could end most source address spoofing. Ingress filtering is one of the simplest technologies, one embodied in an RFC back in 1998, but generally not implemented even today.

Stopping attacks that don't use spoofed source addresses is much harder, even at the source. If the botnet owner decides to launch an attack that uses spidering of a Web site, just having thousands of clients attempting to walk your Web directories will, in itself, be a devastating attack (for most servers). And this attack uses legitimate appearing traffic, not something that an ISP would be able to filter out, if the ISP even noticed.

The core routers appear to be a logical place to stop floods. And while there is some research into this, as well as some working examples, the Internet is a loose federation of networks, and the companies that control the core are competitors. Expecting these companies to cooperate is expecting a lot. You might think that it would be in the interest of the owners of core routers to reduce floods, but the normal traffic seen by their routers is a flood, and the noise of extra traffic gets buried in the background.

That leaves defenses close to the target—your own site. Like just about everything in security, it comes down

to you protecting your own site. The authors suggest many things, including looking for network choke-points, having excessive bandwidth, adding more servers when the load is heavy, and installing patches (some DoS relies on buggy software). But you should also be prepared for a DoS attack. You need to be able to monitor and analyze network traffic at the edge of your networks. Monitoring implies that you have practiced doing this and can easily capture this information and know how to interpret it.

With this information in hand, you can communicate your plight accurately to your upstream ISP, who should be willing to install temporary filters for you. The ISP might even want to communicate with its own providers, pushing back the attack even further.

I will not attempt to duplicate the information in the Internet DoS book here. It has sections appropriate for managers, as well as more technical chapters. I was pleased with the clear and logical prose, even as I was often depressed by the logical implications.

The Internet, as it is designed, accepts any traffic, as long as it complies with minimum standards (a functional IP header). It is fruitless to hope that there are any easy solutions in sight. And that certainly includes solutions that suggest revising IP to defeat DDoS. For the most part, the Internet just works. DDoS and spam are certainly enormous nuisances, but not ones that will by themselves destroy the greatest network ever.

But they sure do make me wish that we had better tools for combating these attacks.

REFERENCES

[1] Jelena Mirkovic, Sven Dietrich, David Dittrich, and Peter Reiher, *Internet Denial of Service* (Prentice Hall, 2005), 372 pp.

[2] Honeynet Project, "Know Your Enemy: Tracking Botnets," http://www.honeynet.org/papers/bots/.

[3] I picked out one variant of Agobot. There are many others: http://www.sophos.com/virusinfo/analyses/w32agobotli.html.

[4] David Brumley, "Tracking Hackers on IRC," *;login:*, http://www.usenix.org/publications/login/1999-11/features/hackers.html.

[5] Honeynet Project, "Know Your Enemy: GenII Honeynets," http://www.honeynet.org/papers/gen2/index.html.

[6] David Moore, Geoffrey Voelker, and Stefan Savage, "Inferring Internet Denial of Service Activity," http://www.caida.org/outreach/papers/2001/BackScatter/.

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

APPLE I REPLICA CREATION: BACK TO THE GARAGE

Tom Owad

Syngress, 2005, 1-931836-40-X, 359 pp. + CD.

THE ART OF COMPUTER VIRUS RESEARCH AND DEFENSE

Peter Szor

Symantec Press/Addison-Wesley, 2005, 0-321-30454-3, 741 pp.

C# PRECISELY

Peter Sestoft and Henrik I. Hansen

The MIT Press, 2004, 0-262-69317-8, 214 pp.

CLASSIC SHELL SCRIPTING

Arnold Robbins and Nelson H.F. Beebe

O'Reilly, 2005, 0-596-00595-4, 560 pp.

HIBERNATE: A J2EE DEVELOPER'S GUIDE

Will Iverson

Addison-Wesley, 2005, 0-321-26819-9, 371 pp.

JAKARTA STRUTS COOKBOOK

Bill Siggelkow

O'Reilly, 2005, 0-596-00771-X, 533 pp.

JOTD: THE WORLD'S GREATEST COMPUTER JOKE BOOK

Hershel Remer ("Rabbs")

Rabbs Publishing (rabbs.com), 2004, 0-615-12449-6, 104 pp.

LINUX DEVICE DRIVERS, 3RD ED.

Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman

O'Reilly, 2005, 0-596-00590-3, 633 pp.

LINUX NETWORK ADMINISTRATOR'S GUIDE, 3RD ED.

Tony Bautts, Terry Dawson, and Gregor N. Purdy

O'Reilly, 2005, 0-596-00548-2, 360 pp.

LINUX QUICK FIX NOTEBOOK

Peter Harrison

Prentice Hall PTR, 2005, 0-13-186150-6, 696 pp.

LINUX SERVER SECURITY, 2ND ED.

Michael D. Bauer

O'Reilly, 2005, 0-596-00670-5, 539 pp.

FEATURED TITLE: THE ART OF COMPUTER VIRUS RESEARCH AND DEFENSE

The Art of Computer Virus Research and Defense, by Peter Szor, is a meticulously researched treatment of viruses, worms, and other types of malicious self-propagating programs, both as entities in themselves and in the context of administering real-world computer systems. The book treats its subjects at an excellent level of detail.

The book's first half provides a very up-to-date description of the ways that viruses and worms function. It includes a thorough history of the general topic as well as a study of attacker strategies and their evolution over time. The second half of the book focuses on responses to them. It covers both infection prevention and post-attack disinfection, including postmortem analysis of the code (in terms of how it the code operates and the obfuscation techniques that it employs).

This book is very well written and is interesting to read. Like all good security works, it manages to get across how the bad guys think and operate in ways that are useful for the good guys but without providing any help to black hat wannabes. This book will be very useful for system administrators and other computer security professionals, as well as computer scientists interested in this area of research. It also contains information of interest to programmers concerned about writing secure code.

FOUR PLANETS IN THE PROGRAMMING UNIVERSE

This month brings us four programming titles, each focusing on a specific, specialized programming context.

Classic Shell Scripting, by Arnold Robbins and Nelson H.F. Beebe, is an excellent book in the classic tradition of O'Reilly & Associates. It is an accurate, comprehensive treatment of writing shell scripts using modern Bourne-style shells. The

authors explicitly model their work after the Kerninghan and Plauger classic *Software Tools* volumes, and one could obviously not ask for a better approach. The book also provides an excellent introduction/reference for regular expressions, sed, awk, and many other standard UNIX tools. Although the authors occasionally go a bit too far—they truly believe that the shell is the best solution for virtually any programming problem—this book is nevertheless the definitive work on shell scripting.

Bill Siggelkow's *Jakarta Struts Cookbook* provides useful information and a plethora of helpful examples for programmers creating Web applications with Java. After some preliminary information about installing and configuring Struts, the book contains a well chosen and organized collection of code excerpts. The examples are structured as problems (goals) and solutions, a technique which results in well-planned examples (rather than merely a somewhat random collection of them). The solutions themselves usually cover not only the specific task at hand but also several variations. All in all, this is one of the very best volumes in the O'Reilly *Cookbook* series.

C# Precisely, by Peter Sestoft and Henrik I. Hansen, is a reference for the new version 2.0 of C# (Microsoft's Java-like object-oriented programming language, designed for use with its .NET Framework). The book focuses on the programming language itself, choosing to ignore most of the .NET class libraries. Language features are discussed on lefthand pages, with related examples on the corresponding righthand pages. The book will be found to be both readable and useful for programmers who use C#.

Hibernate: A J2EE Developer's Guide, by Will Iverson, provides comprehensive coverage of Hibernate, a widely used package designed to automate the process of mapping relational database structures to ordinary Java objects (typically saving a lot of programming effort and development time over using JDBC). Unlike other works on Hibernate, this book is organized around building a real application from the ground up. It begins with discussions of creating schema and mappings, the essential infrastructure required by every application. Later chapters cover the resulting Java classes, queries within the Hibernate framework, transactions, application performance, and so on. I find this organizational structure to be both logical and natural if the ultimate goal is to create real-world Java applications.

A DIFFERENT APPROACH TO LINUX SYSTEM ADMINISTRATION

The *Linux Quick Fix Notebook*, by Peter Harrison, takes an unusual approach to Linux system administration. Its audience is system administrators who want to configure a Linux system for use as a Web server (although much of its discussion would also apply to a system designed to be a file server). It is designed to be useful to both Linux users moving to this particular administration task and Windows Web server administrators who are moving to Linux. The book uses the command line for every configuration task in order to sidestep the Linux distribution quagmire, a clever tactic in my opinion. The work is procedure-oriented, avoiding most conceptual discussions in favor of focusing on how to get specific tasks done. Nevertheless, the book provides sufficient and accurate information which will enable members of its target audience to successfully configure a Linux Web server.

NEW EDITIONS OF LINUX CLASSICS

New editions of several Linux references are now available. The third edition of *Linux Device Drivers*, by Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman, updates that work for the 2.6.10 kernel. The book remains the definitive treatment of this topic, and it provides an excellent means for an experienced programmer to write a device driver for the first time.

The third edition of the *Linux Network Administrator's Guide*, by Tony Bautts, Terry Dawson, and Gregor N. Purdy, is an extensive rewrite. The new version removes discussions of outdated technologies (e.g., IPX, uucp, Internet newsgroups) and adds brief overviews of Apache, Samba, LDAP, IMAP, and wireless networking. Existing discussions have also been updated to cover IPv6 and iptables (instead of earlier tools, in the latter case).

The second edition of *Linux Server Security*, by Michael D. Bauer, is a revision of *Building Secure Servers with Linux* and is probably the least extensive revision here. It adds coverage of LDAP and databases to the previous work.

WAY, WAY OFF THE BEATEN TRACK

I'll close this column with a brief look at two quite eccentric works. *JOTD: The World's Greatest Computer Joke Book*, by Hershel Remer (a.k.a. UnixRabbi, a.k.a. Rabbs), does not live up to the claim in its subtitle, but it did provide me with a fair amount of mild humor. I suspect that readers with a greater tolerance for gender and ethnic stereotype-based humor and Microsoft bashing will find it quite amusing.

I have a friend who recently came across an IBM mainframe emulator on the Internet and thus had an urgent need for books on JCL (of which there are, unbelievably, some actually still in print). If that seems cool rather than bizarre to you, then Tom Owad's *Apple I Replica Creation: Back to the Garage* may be of interest. The book takes you through the process of building an Apple I replica and then programming it. It comes with a copy of the McCAD EDS-SE400 integrated design software. Happy building.

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*

VALE, ROB, ATQUE AVE, RIK

Ellie Young
USENIX Executive Director

In the March/April 1992 issue of this magazine (then still a newsletter), an announcement appeared on page 3 entitled "Welcome to the New Improved *;login:*." The piece, signed "Rob," marked the beginning of *;login:* as we know it today—conference reports, feature articles, "how-to's," and an expanded book review section. Since that day, editor Rob Kolstad, seeking out and working with contributors, has continually improved *;login:*'s quality and interest for you, the membership.

Now it is time for another change. Beginning with the August issue, Rik Farrow is taking over the responsibility so well carried out by Rob. Kolstad transformed *;login:* into a magazine that is now considered by the membership to be one of the most valuable benefits that USENIX provides. We are profoundly grateful to him for his hard work, goodwill, and effort over the 15 years of his tenure. Rob is not going away: he continues to be our executive director of SAGE, advisor to LISA conference organizers, and typesetter of the proceedings for LISA, and to run the USENIX-sponsored high school Computing Olympiad, to host the game shows at our events, and more.

We have been fortunate in having Rob serve for so many years as the prime mover of *;login:*. We are doubly fortunate that he is still engaged in the USENIX enterprise.

Rik Farrow, as many already know, has long been involved in *;login:*, both as the author of a regular column, "Musings," and, since 1996, as the editor responsible for the annual special issue on Security. Rik has considerable experience as an editor, both as editor of the SAGE Short Topics series and as technical editor of *UNIXWorld*

from 1989 to 1994, and as a writer and teacher on various security topics.

It is once again our good fortune that Rik is willing to take the helm at *;login:*. We look forward to his long and successful tenure!

SUMMARY OF USENIX BOARD OF DIRECTORS MEETINGS, DECEMBER 2004–APRIL 2005

Tara Mulligan
USENIX Member Services Manager

MEMBERSHIP

The Board voted to increase Institutional member benefits as follows:

Educational: Will receive up to 2 additional subscriptions to *;login:*.

Corporate: Will receive up to 4 additional subscriptions to *;login:*; will be provided with five (5) member-priced conference registrations during the membership term; will receive multiple-employee discount registrations to allow more staff to attend USENIX conferences; and will be listed on a page linked from our Membership portal page during the membership term.

Supporting: Will receive up to 4 additional subscriptions to*;login:*.

The new benefits will be implemented over the next few months. If you are interested in upgrading your account or in learning more about our classes of membership, please see www.usenix.org/membership or contact us at membership@usenix.org.

CONFERENCES

The USENIX Annual Technical Conference will be moved back into the early summer timeframe in 2006. It will also be reformatted to address current issues in advanced computing systems. Please check your USENIX news email messages and conference announcements for further developments.

The Board agreed to be a sponsor of CodeCon 2005, which was held in San Francisco in February.

USENIX will make a $20,000 donation to Stichting SANE for the SANE 2006 conference and will offer a guarantee in the event that there is a deficit.

COMMITTEES

The Board will create a subcommittee chaired by Matt Blaze to look into fraudulent/dual paper submissions to conferences.

NEXT MEETING

The next regular meeting of the Board of Directors will be held on Tuesday, August 2, 2005, at the USENIX Security Symposium in Baltimore, MD.

ANNUAL AWARDS

USENIX LIFETIME ACHIEVEMENT (FLAME) AWARD WINNER 2005: MICHAEL STONEBRAKER

Board President Michael B. Jones presenting the Flame Award

Dr. Michael Stonebraker has been a leading database, operating systems, and expert systems designer, both as an academic and as an entrepreneur, for over thirty years.

He is well known for his work in developing both the INGRES and POSTGRES database systems under a freely distributable BSD license, then going on to form companies (Ingres Corporation and Illustra Information Technologies, Inc.) to market them as commercial products. He also initiated the Mariposa project, which became the basis of another company called Cohera, later sold to PeopleSoft. All three of these projects were developed at the University of California, Berkeley, where Dr. Stonebraker served as a professor of computer science for 25 years.

Currently Dr. Stonebraker is an adjunct professor of computer science at M.I.T., where he has helped build a stream processing engine, Aurora. In 2003 he founded StreamBase Systems to market this technology, with himself as CTO.

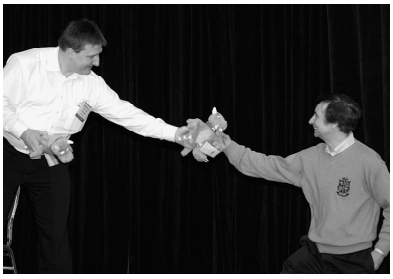
He has authored and co-authored scores of research papers on database technology, operating systems design, and expert systems. He has been active in the ACM Special Interest Group on Management of Data (SIGMOD) both as a member and a leader.

He has a B.S. from Princeton (1965) and an M.S. (1967) and a Ph.D. (1971) from the University of Michigan.

Dr. Stonebraker has also received several other awards, including the IEEE John von Neumann Medal in 2005, the ACM Software System Award in 1988, and the ACM SIGMOD Innovations Award in 1994. He was elected to the National Academy of Engineering in 1998.

USENIX STUG (SOFTWARE TOOLS USER GROUP) AWARD WINNERS 2005: MATTHIAS ETTRICH AND MIGUEL DE ICAZA FOR KDE AND GNOME

The STUG award recognizes significant contributions to the community that reflect the spirit and character demonstrated by those who came together in the Software Tools User Group (STUG). Recipients of the annual STUG award



A KDE mascot goes to GNOME

conspicuously exhibit a contribution to the reusable code-base available to all and/or the provision of a significant enabling technology to users in a widely available form. The UNIX Command Line User Interface (CLI), while widely recognized as being efficient, has often been attacked by non-UNIX users as not user-friendly. In response, many GUIs have been added to UNIX over the years, but most were generally considered inferior to non-UNIX GUIs.

In October of 1996 and August of 1997, two projects were started to produce desktops that were easy to use, adhered to traditional UNIX philosophies, and gave access to all of the underlying features of the CLI.

While these desktops competed with each other, they also lent strength to each other and have now produced a range of applications that we collectively call KDE and GNOME. These applications have eased implementations of the UNIX operating system in the non-technical marketplace. Most important, by embracing the concepts of free and open source software, these two desktop projects offered freely distributed code, which allowed any distribution or software developer to utilize these graphical features.

The USENIX Association would like to recognize both of these groups for creating a very portable set of libraries, tools, and applications.

YEARS AND YEARS AGO

Peter H. Salus
peter@usenix.org

This issue of *;login:* is "volume 30, number 3." But, of course, it isn't.

The purple ditto'ed sheets created by Mel Ferentz in July 1975 were headed "UNIX NEWS." It was years later, following a threatening letter from a lawyer at Western Electric, that the name of the newsletter was changed.

My copy of "UNIX NEWS," Number 1, is dated July 25, 1975, and contains the notation "Circulation 37."

Dennis Ritchie and Ken Thompson delivered the first UNIX paper in October 1973. Lou Katz and Reidar Bornholdt convened the first "UNIX Users" meeting on May 15, 1974. The UNIX paper appeared in the July 1974 *CACM*, and Mel sent notices to 37 institutions of the arrival of V6.

And a meeting, organized by Lou, Reidar, and Mel and hosted at CUNY by Ira Fuchs, was held on June 18. Mel described it in that first newsletter:

"The meeting on June 18 at City University of New York was attended by over 40 people from 20 institutions. Each institution described briefly its function and idiosyncrasies. . . . There was unanimous sentiment for keeping the user's group and its newsletter as informal as possible."

Over the decades both the "user's group" (now USENIX) and the newsletter (now *;login:*) have shed a good deal of that informality. But there's a lot of the cohesive spirit still there 30 years on.

;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
RIDE ALONG ENCLOSED